ABSTRACT

A DISCOVERY AND ANALYSIS OF INFLUENCING FACTORS OF PAIR PROGRAMMING

by
Kyungsub Steve Choi

The exploration into the underlying psychosocial links of pair programming, a new and unorthodox programming paradigm in which two programmers share one keyboard and monitor during real-time programming sessions, is undertaken. These complex psychosocial relationships, along with cognitive process exchanges, ultimately mold the programming output as well as determine the level of communication, satisfaction, confidence and compatibility. Laying the framework for this research, a through review of traditional and contemporary paradigms with a special focus on their limitations and a list of current software development problems are presented. Next, a detailed summary of pair programming and related agile software paradigms, such as extreme programming, which lists pair programming as one of its twelve principles, is given. From earlier pair programming studies, a number of programming benefits have been unveiled and these are listed and discussed. However, a lack of formal studies pertaining to the psychosocial aspects of pair programming exists. Given this void, a field survey is administered to a group of professional programmers and a resulting list of influencing factors on pair programming emerges. From the list, the most popular factor, personality, and two other factors, communication and gender, have been selected in order to study their impact on pair programming product outcome and the level of communication, satisfaction, confidence and compatibility. An experiment focusing on these factors is designed and implemented. From the experimental findings, the personality of the two

partners in pair programming is found to have a significant impact on the pair programming output. Also, it is discovered that same gender pairs exhibited an unusually high level of communication, satisfaction and compatibility between each other, especially among female-female pairs. A detailed statistical experiment result based on research hypotheses is reported.

A DISCOVERY AND ANALYSIS OF INFLUENCING FACTORS OF PAIR PROGRAMMING

by
Kyungsub Steve Choi

A Dissertation
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy in Information Systems

Department of Information Systems

May 2004

APPROVAL PAGE

A DISCOVERY AND ANALYSIS OF INFLUENCING FACTORS OF PAIR PROGRAMMING

Kyungsub Steve Choi

Dr. Fadi P. Deek, Dissertation Advisor — Date
Professor of Information Systems, NJIT

Dr. Il Im, Dissertation Co-Advisor — Date
Assistant Professor of Information Systems, NJIT

Dr. Murray Turoff, Committee Member — Date
Distinguished Professor of Information Systems, NJIT

Dr. Bartel Van De Walle, Committee Member — Date
Assistant Professor of Information Systems and Management,
Tilburg University

Dr. Michael Hinchey, Committee Member — Date
Director of the Software Engineering Laboratory
at NASA Goddard Space Flight Center

BIOGRAPHICAL SKETCH

Author: Kyungsub Steve Choi

Degree: Doctor of Philosophy

Date: May 2004

Undergraduate and Graduate Education:

- Doctor of Philosophy in Information Systems,
  New Jersey Institute of Technology, Newark, NJ, 2004

- Master of Science in Management Information Systems,
  New Jersey Institute of Technology, Newark, NJ, 1998

- Bachelor of Science in Management Information Systems,
  New Jersey Institute of Technology, Newark, NJ, 1997

- Bachelor of Arts in Chemistry and Economics,
  Rutgers College, Rutgers University, New Brunswick, NJ, 1992

Major: Information Systems

Presentations and Publications:

Choi, K. S., and Deek, F.P., "Extreme Programming, Too Extreme?," The Proceedings Of the International Conference on Software Engineering Research and Practice 2002 (SERP'02), Las Vegas, USA, 2002.

Choi, K. S., "Imposing Computer Mediated Communication Theories on Virtual Reality", The International Conference on Information Technology: Research and Education, (ITRE2003) August 10-13, 2003, Newark, New Jersey, USA.

Choi, K. S., "A Discovery and Analysis of Influencing Factors of Pair Programming," The Software Engineering Research Symposium; NJIT and NASA, January 29, 2003. Newark, New Jersey, USA.

ACKNOWLEDGEMENT

Looking back in retrospect, the completion this work was not just the result of my efforts but also those of many others. I am deeply indebted to numerous mentors and friends who have helped make this accomplishment possible. I would first like to begin by expressing my sincere gratitude to my advisor, Dr. Fadi P. Deek. He warmly welcomed me under his guidance several years ago and has provided me with an overwhelming amount of guidance and support ever since. He has not only allowed me to develop and refine my ideas and skills in an academic research environment, but he has also been a good friend who has provided the much needed encouragement and emotional support which helped me to complete such a long and difficult journey.

I would also like to thank Dr. Il Im, who is my co-advisor. His advice and help during the creation of the statistical report and the final write up were invaluable and this research effort could not have been completed without him. Despite countless emails and office visits, Dr. Im always welcomed me with a smile and offered great insights during the final stage of this work.

I am also grateful for my committee members, Dr. Murray Turoff, Dr. Bartel Van de Walle, and Dr. Michael Hinchey, for all of their valuable time and words. By sharing their expert advices, each member has helped me refine and polish this paper.

Also, the completion of my postgraduate career would not have been possible without financial assistance. For that, I thank NJ I-Tower, who funded my research assistantship.

I also must express my deep gratitude to a number of professionals who helped me immensely in this effort. First, I sincerely thank Dr. Brian H. Spitzberg of School of

Communication, San Diego State University, who had graciously afforded the use of his communication skill measurement instruments to a PhD student who he has never met. Also, I would like to express my heartfelt thanks to Dr. Bill Anderson for letting me use his valuable instrument and to Dr. Ulla Bunz of School of Communication, Information and Library Studies, Rutgers University for her kind suggestions and the introduction to Dr. Spitzberg. I would also like to thank several other trusted friends who have contributed their time as either a problem design contributor, an experiment subject supplier, a experiment space provider, or a judge; they are Prof. Maura Deek, Dr. Robert Friedman, Dr. Joanna DeFranco-Tommarello, Prof. Ted Nicholson, Prof. Morty Kwestel, Tom Cohn, Vikas Patel and Anika Raut. I also thank my colleagues, professors, and the NJIT IS department for their guidance and support.

Finally, I would be remiss if I didn't mention a list of friends and family who have offered their valuable time and friendship during this journey. Sincere thanks goes out to James Yusko for his help in my writing and friendship, to Yoo-Nam Kim who not only provided his friendship, but also his home when I needed one, and to Yong-Min Han for his sincere friendship during some very difficult times. I am always grateful for my two brothers, Kyung-Jo and Kyung-Ho, and my sister, Kyung-Mi. I love you all.

This journey was truly a struggle; a struggle that I shall remember for the rest of my life and one that served to strengthen my character and resolve. There were three occasions when I seriously contemplated quitting, but because of the love of my family and friends, I decided to continue. My last thanks goes to the sport of marathon running, which has been a great outlet for me and something that I consider to be a lifetime teacher and mentor.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

TABLE OF CONTENTS
(Continued)

TABLE OF CONTENTS
(Continued)

LIST OF TABLES

LIST OF TABLES
(Continued)

LIST OF TABLES
(Continued)

LIST OF TABLES
(Continued)

LIST OF FIGURES

LIST OF FIGURES
(Continued)

CHAPTER 1

INTRODUCTION

Creating an efficient, risk-free, and prudent software development process has always been the constant pursuit of software engineers (Boehm, 1976; Royce, 1987; Jones, 1994; Highsmith, 2000; Cockburn, 2000; Eljabiri and Deek, 2001; Erdogmus, et al., 2002; Boehm and Turner, 2003). However, despite many efforts, it is widely agreed that the results are still far from satisfactory (Boehm and Basili, 2001). Throughout history, a number of software development problems have been exposed which lead to a birth of one software process paradigm after another (Royce, 1987; Boehm, 1988; Smith, 1991; Prowell, et al., 1999; Cockburn, 2001). The problems may be categorized into two groups. One group pertains to problems that arise from software process paradigms being unable to meet the needs of the business drivers (Cockburn, 2000; Eljabiri and Deek, 2001) and the other group includes problems that result from social inertia (Keen, 1981; Markus, 1983; Yourdon, 1997), which plays a large role in impeding the implementations. Along with these problems, the wide spectrum of various situations and projects also demands flexibility and availability from the software process paradigms (Davis, et al., 1988; Pressman, 1997a; Pressman, 1997b).

Just as with many other elements in the volatile business terrain (Goldman, et al., 1995), the software process development paradigms that ultimately produce the goods for business must be flexible to any given requirements (Baskerville and Heje, 2001b). The continuous cycle of shedding the old and fitting the new, exponential growth of e-commerce, teamwork with customers and balance between theory and practice are all becoming relevant concepts and influencing factors on the new standard (Boehm, 2002a).

But of all of these concepts, the most significant concept may be the 'time to market' factor (El Sawy, 2001; Cusumano, and Yoffice, 1998; Cusumano, and Yoffice, 1999; Stalk and Hout, 1990). Originating from the business ecosystem, it's definitely found a role in software process paradigms (Cusumano, and Yoffice, 1999; Baskerville and Pries-Heje, 2001a; Baskerville, et al., 2003). More than any other tools or mechanics of a software process paradigm, the time factor is becoming the primary concern in software process paradigms (Baskerville and Pries-Heje, 2001a; Baskerville and Pries-Heje, 2001b; Highsmith and Cockburn, 2001). It is as equally significant as the level of quality of the finished product. Cumulatively, the tools of traditional paradigms are becoming inadequate in the face of this new tighter standard. All of these problems coupled with continuously growing requirements have the software community desiring other options.

As one of the possible solutions, a new movement has emerged. The "agile" movement (www.agilemanifesto.org; Cockburn and Highsmith, 2001; Highsmith and Cockburn, 2001; Boehm, 2002b) defies the conventional logic with a set of unorthodox tools (Beck, 2000; Cockburn, 2001; Abrahamsson, et al., 2003; McCauley, 2001). Among the agile software process paradigms is a bold new paradigm known as extreme programming (XP) (Beck, 2000; Beck and Fowler, 2001). Included among XP's twelve key principles is its implementation of pair programming (PP). Briefly, PP can be defined as a new programming style in which two individuals physically or virtually share one terminal and keyboard for collaborative programming purposes in real-time (Williams and Kessler, 2002). Pair Programming presents many interesting research questions such as what are PP's changes on productivity, quality, and satisfaction when compared to that of the conventional one-man programming concept (Williams, et al.,

2000; Williams, 2000). There has been some research on XP and PP and their implications (Williams, 2000; Williams and Upchurch, 2001, Williams and Kessler, 2000; Williams, et al., 2000), but there is a dearth of study pertaining to the human role in XP and PP, an area that may be the most critical and important in PP. Therefore new research that focuses more on the "engine" of PP, the human factor, is needed (Cockburn and Highsmith, 2001).

This research addresses the human attributes or the psychosocial dimension of PP. One primary research goal is to unearth the underlying psychosocial ties between the two individuals of PP and further analyze the selected ties for better enhancement and optimization of PP. In order to lay a framework for this research, a few preliminary objectives have to be addressed. First, an attempt is made to alleviate confusions that stem from the software process terms. And a visit to a list of relevant terms and their corresponding definitions is arranged. A review of software process paradigms, both traditional and contemporary is also presented. Only after a realization of the limitations of the paradigms can a constructive analysis take a place. In addition to addressing the paradigm limitations, some of today's software development problems are also examined. By accurately assessing the contours of these problems, a keen vision for new paradigms may develop. An overall implication of contemporary software process paradigms needs to be discussed and also evaluate how this new concept is remedying the problems.

In the second part of the research, PP experiment is focused. A field survey given to a group of professional programmers with PP experience reveals a set of PP influencing factors. The influencing factors would be further evaluated as they may illustrate the boundaries of underlying psychosocial dimension and also serve as a

springboard for research questions. As an additional analysis, some of the influencing factors may be further examined in order to determine their validity and degree of strength. An analysis of PP teams versus one-man teams is also presented. The experimental findings may provide a set of guidelines in dealing with those influencing factors and how one can manipulate them in order to achieve a higher PP output.

CHAPTER 2

REVIEW OF SOFTWARE PROCESS PARADIGMS

2.1 Definitions Revisited

2.1.1 Software Process Model Defined

There have been many grandeur review literatures about the software process models and methods (Pressman, 1997a; Sommerville, 2000; Behforooz & Hudson, 1996; Von Mayrhauser, 1990; Pressman, 1997b). From the "Grand daddy" Waterfall (Royce, 1987) model to the latest web-based model (Aoyama, 1998), the history of the software process models marks the change of time in software engineering. The internal needs, such as the model's own limitations, and the external factors, such as the factors of the business ecosystem, have shaped and molded the history. From the early days of the software crisis to today's race against time, the software community truly has elevated the knowledge and competency in software production. Today, a numerous names and synonyms for the models and the methods exist. Each is serving the exact need for a particular ecosystem, a particular purpose, and a particular group of people. It's up to the users to decide the "where and the how" for the usage, therefore the variety.

From the beginning, selecting a particular software process model for a project is a gray area for the practitioners (Cockburn, 2000). In the software engineering realm, picking out an ideal software process model or a method is not a scientific and logical process. One's past personal work experience has a lot to do with the decision of which model and method to use and why. With the efforts to address this area (Boehm & Belz, 1990; Eljabiri & Deek, 2001; Sommerville, 2000), there is not an industry-accepted norm in reference to this decision-making. It is perceived that many software development problems occur as a

result of the mismatch between the process model used by the project and the project's real-world process drivers.

For the general software process models review, the prominent models are selectively presented. They are: 1) linear sequential or Waterfall model, 2) prototyping model, 3) spiral model, and 4) COTS model. All available models and their attributes are initially classified and categorized. From this process one would surmise that the linear sequential and prototyping models are the two "root" models from which many others models are based. The spiral and COTS models are prominent models in that their positions are at large among the models. Because of what each model brings and contributes, the demand for each model is rising. With its salient feature of risk analysis, the spiral model not only accommodates the business perspective, but the strong quality emphasis in its core process as well (Boehm, 1988). The rising popularity of COTS is startling, but it will probably continue in the foreseeable future. Today, it is common to encounter systems that are either full CBS or a part CBS with some added in-house components. The spiral and COTS models are explained with their attributes in detail and presented with some associated limitations.

No methods (sometimes referred to as techniques), are covered in this paper. Although methods are mentioned briefly and sporadically thorough out the paper, no designated full section is devoted to just the methods. There is no specific reason behind this omission, except that this paper's scope does not include a detailed section for the methods.

Before the review of the named models and methods, it would be better served to review the nature of the software process models, such as names, terms, synonyms, stages, processes, and other associates. Many names, terms, and synonyms of the models

can be better "traffic controlled". It can be a bit confusing when such a situation exists. Starting with the definitions, an attempt is made to address this question.

2.1.2 Model vs. Method

The confusing experience of the names and the terms is not surprising. Basically, the names, terms, and definitions are the views of the originators. It's wholesome to have different perspectives on an item among the researchers. Also, one may argue for the little value of this discussion, but the confusion still exists in some and maybe in many. Having said that, whatever the paper's concluding "verdicts" may be on the matters, they are only the views of this paper. The following is the first group of terms that is to be discussed for their appropriateness: 'lifecycle model' (Behforooz & Hudson, 1996), 'paradigm' (Pressman, 1997a), 'process model,' (Sommerville, 2000; Kellner & Hansen, 1988), 'method,' and 'technique'. Largely, they are sorted into two different groups; one group consists of the 'lifecycle model,' 'paradigm,' and 'process model', which have definitions that are similar to each other, and the other group consist of 'method' and 'technique,' which also have similar definitions. The first group is referring to the complete process of the software development, from the beginning to the end, the requirement solicitation stage to the placement in the production stage. The other group is referring to an effective tool or an approach that is interjected into a certain stage of the software development. It's clear that 'method' and 'technique' are not the terms referring to the complete process of the software development. Here are some definitions for the first group:

"Software Process Model- *A methodology that encompasses a representation approach, comprehensive analysis capabilities, and the capability to make predictions regarding the effects of changes to a process*". (Kellner & Hansen, 1988)

"System Development Life Cycle – *A framework composed of a sequence of distinct steps or phases in the development of a system*". (Anderson & Dorfman, 1991)

"Software Life Cycle - *The period of time that begins when a software product is conceived and ends when the software is no longer available for use. The life cycle typically includes a concept phase, requirements phase, design phase, implementation phase, test phase, installation and checkout phase, operation and maintenance phase, and sometimes, retirement phase* ". (IEEE Std 610.12-1990, 1990)

Like the IEEE's definition, the complete process refers to the distinct stages and it encompasses from the requirement gathering, designing, coding, testing, implementing, maintaining, and retiring. The complete process is a "cradle to tomb" approach. Many forget to include the maintenance stage and the retirement stage in the complete process. There are equally important activities involved in those two stages. Based on the literature definitions, 'lifecycle model', 'paradigm', and 'process model' are interchangeable terms in the software engineering context. For the purpose of this paper, the term 'model' for 'process model' is used. In summary, a software process model

eans the "cradle to tomb" process in the software engineering context. Any model that did not exhibit the " cradle to tomb" concept was excluded from the model review list and labeled as a method.

A method is, in contrast to a model, an effective catalyst for a stage or stages of the software process model. One quick visual example is the prototyping utilization during the requirement solicitation. The prototyping method is an extremely useful tool during the requirement gathering stage, however it's only a method or a technique. A clear separation exists, as the software process model is a lengthy process that encompasses the stages from the requirement to the retirement, and a software process method or technique is an effective tool that assists a software process model in enhancing or bettering a stage or stages.

2.1.3 Software vs. System

Terms such as "xxx system" or "xxx software" are often heard when it actually is referring to the system. Without any hesitation, both the speaker and the listener interchangeably use either term. In fact, "xxx software" is used more often when it should be "xxx system". The confusion hits when a need to refer to a system or a software application arises. One might say, "You mean the software? Right?" Without revealing the literal definitions of each, it's clear that the two are not the same. With software and hardware comprising the final system, why does the term "xxx software" is used more often and also use the terms interchangeably? Here are the possible explanations:

1) Software is the only visible part and all of the visible parts to the eyes of the end users.

2) It's a natural inclination to refer by the most "used" component

3) Our minds experience the software, not the hardware

4) Technically, the "little" hardware presence is incomparable to that of the software's

There are published works regarding the nature of 'software' and 'system' that speak of the differences (Thayer, 1997; Forsberg & Mooz, 1996; Boehm, 2000). Sometimes it's necessary to clear up the difference in a document. As the phenomena of the importance of software and hardware are headed in opposite directions, this unclearness may stay for quite a while. This is also relative to the increasing software engineering presence.

2.1.4 The Stages of Model

As the definitions of the models put all of them in the same path as the methods, the internal intricacy and the structure of the stages of models make the models differ from each other. Coming from the waterfall model, the generic order of the stages is requirement gathering, designing, coding, testing, implementing, and maintaining (Royce, 1987). The most vivid difference among the software process models has to do with the stages. For some models, the order of the stages is different by the addition of a new stage. The Evolutionary Incremental prototyping model and the Spiral model each present a unique process different from that of the waterfall model. The Spiral model has the risk analysis stage in every round cycle (Behforooz & Hudson, 1996). Department of Defense (DoD) model has unit testing, integration testing and user acceptance testing integrated with other stages (Boehm, 1988; Prowell et. al., 1999). The following Table is a brief list of synonyms of the stages:

Table 2.1 Synonyms of Software Process Stages

Most common name	*Other names*
Requirement gathering	User requirement Requirement engineering Requirement solicitation Requirement definition
Designing	Developing Planning
Coding	Implementation
Implementation	Acceptance testing Validation Verification

Just as the stage names vary, the document or documents that generate from each stage are equally diverse. The role of documents is very important, as no system is a system without its associated development documents. However, this is always changing (Cockburn, 2001). Some industries, such as U.S. pharmaceuticals, are legally bound to a generation of a set of clear, crisp, and error free development documents. The synonym confusion may also exist in the document names. The following is a partial collection of the development documents:

Table 2.2 Software Development Documents

Stage	*Document name*
Requirement	User Requirement Specification System Requirement Specification Software Requirement Specification
Designing	Design Specification Concept of Operations document
Coding	Technical specification
Testing	Verification and Validation Report
Implementing	Implementation Report

A different list of the stages may generate a different list of documents. Each document confirms the validity and integrity of the corresponding stage. The document or documents of a stage assure the completeness of the stage. However, due to the document's inherent nature, continuous updates and modifications to the documents are inevitable. Better document control and management has always been an object of study (Budlong, and Stanko, 1993; Han 1994). The documents can also become a fine tool in the development process (Lutsky, 1995; Borstler, J.and Janning, 1992). Each document serves its purpose by providing an accurate account and record of what had taken place in that particular stage, thereby software audit, inspection, and reverse engineering exist.

2.2 Traditional Paradigms

One of the reasons for the availability of many models is the dynamic and fast evolving business drivers and environment. Regardless of what type or kind of model, one hears the same voice; understand the requirements, design the solutions, code the solutions, test the solutions, and the implement. The maintenance stage is added as an ongoing process. In searching through and reviewing the various models, a fact reveals that some model names are just synonyms for a particular model. Also, the different degrees of subtle differences in the descriptions of the models were noted by the different literatures. The different purpose, context, and usage may have been attributed to these differences. One note on the prototyping is that when it's evolutionary or incremental then it's a model, but if it's experimental or throwaway then it's a method or technique.

2.2.1 Linear Sequential

• Description

To all software engineering minds, this linear sequential model (or Waterfall model) is the icon to what a software process model is. It's the grandfather model to all software process models, and it's the first software process model that every software professionals taught. Although still going strong, today it mainly serves as a rudimentary model for other models to spring off of. It's the first model to present the sequential stages of the software process model. They are the 1) requirements elicitation, 2) software designing, 3) software coding, 4) software testing, 5) implementing, 6) maintaining, and lastly 7) retiring. This has been the bible for all those models that came afterwards. The first notable work on the waterfall model is from Barry Boehm (Boehm,

1976) and W.W. Royce (Royce, 1987). It was the embryonic time of the software ecosystem where no specified development guideline existed and people were more sold to the hardware performance. Hardware was much more expensive then and the research focused on the efficient utilization of hardware resources. Software was only looked as a supporting piece and the human labor was cheap in the spotlight of hardware.

- Application

It's not an overstatement that the linear sequential model was the blue print for all existing models. Because it contains the essential development stages, other models simply took the stages and have arranged the stages differently, added new stages, or modified the stages to suit their needs and purpose. The strongest point of this model is the stability. Structured and pre-defined linear sequence gives security and assurance to the team from the elicitation of requirement stage to the implementation. There's no confusion here as each previous stage must be completed before going onto the next following stage.

Generally, this model is a choice for a large project, such as the redevelopment of a legacy system, and military applications. The key item is that the requirement is known and it will stay as is during the process. Also knowing the risks and potential bugs ahead of time allows the team to deal with them prior to beginning and manage them in the most efficient manner. All the time-consuming fixes and the numerous re-corrections can all be done at once. Given the time favor in this model, the team thinks and executes with fine details. The notable members of the linear sequential model family are the V-shaped model and Department of Defense (DoD) model, or NASA model.

The incentive of the V-shaped model is the accountability of the stages; coding being the pivot point, the earlier stages (requirements, high-level design, low-level design) are accounted for in the later corresponding stages (acceptance test, integration test, unit testing). It's the linear sequential model with accuracy and reliability in mind. The traceability of each defined task to its tests and implementation assures that the task is done and done accordingly. In the unit testing, every small sub-module is tested before it gets to be integrated into the next parent module, hence the error rate is extremely low (Linger and Trammell, 1996).

The DoD model (DoD-Std-2167A), or NASA model, was exclusively created for the military and NASA projects. These projects are in a different context than commercial projects. They absolutely focus on the safety, reliability, and accuracy. Dealing with human lives, a project's failure should not be the cause of it. There is neither the market pressure nor the "moving target" of the requirements. A uniqueness of the DoD model is that the DoD model does not engage in the user requirements elicitation. Rather, they're gathered from the system design documents (Behforooz & Hudson, 1996). Unlike the other models, there is no user involvement in this model. The soldiers and astronauts have a very limited development input, if any at all.

• Limitations

As the software ecosystem matures, the limitations are more profound and revealing. Many of the other models were born out of the linear sequential model's limitations. Among many, the major limitations are: 1) inflexibility, 2) no generosity to the errors, and 3) slowness. The inflexibility doesn't allow one to go back a stage or stages. Because all of the required tasks of a stage are completed in that stage before going on to

the next sequence stage, it's just not feasible to go back. The consequences are overwhelming. This stifles many software professionals because the software development starts from an empty vacuum. The evolutionary model and incremental model were mostly designed to fill this need. The no generosity to the errors limitation attributes greatly to lengthening the project completion time. The ripple effect of the corrections not only slows the project completion but also can bring a disaster to the project. The tradeoffs of the slowness are the accuracy, abundance of various testing, and appropriate safety measures. The linear sequential model is no longer the main model in software development; it's being relegated to a role of reference for all software process models.

2.2.2 Spiral

• Description

Originated by Boehm (1988), the spiral model is a hybrid model that is made of a mixture of the prototyping, evolutionary, and incremental models along with conventional management of evaluation and risk analysis (Boehm and Belz, 1990; Behforooz and Hudson, 1996; Von Mayrhauser, 1990). Named after its 'spiral' process flow, the spiral cycles execute in these four quadrants; they are: 1) determine objectives, alternatives, and constraints 2) evaluative alternatives, and identify and resolve risks 3) develop and verify next-level product and 4) plan next phases and back to the beginning of the cycle again. The key incentive of the spiral model is the risk analysis. It gives the team a process evaluation opportunity in each cycle. It's an opportunity for the team to play devil's advocate weighs pros and cons of next cycle implementation, affirm their process, and consider a possible project withdrawal.

• Application

The spiral model is applied: 1) in projects where the requirements are not uniform, complete, or formal and that they carry certain risks, 2) in projects that seek in eliminating errors and unattractive alternatives early and 3) in projects that may want to add ad-hoc desired features (Boehm, 1988; Von Mayrhauser, 1990). It works similar to the incremental and evolutionary models, in that in each spiral cycle a feature or a set of features are added which allow the requirements to be partially completed or in-progress. In doing this spiral cycle, the team can observe what works and what doesn't even before the complete software project is done. Being able to see the effect of each feature's performance allows the team to adjust or delete the ineffective feature. The spiral model would be a good choice for risky projects that have unproven features and for projects that carry some generous time period.

• Limitation

The limitations are: 1) the project can be cancelled at any time, 2) it's a very time consuming model, and 3) the finished product may not be the product that the requirements expected. Because this is a risk analysis model, the project may be under a larger probing and therefore there is a higher project cancellation risk. This may challenge the model purpose. In every spiral risk analysis, the team meeting can last long and there can be more questions and pondering to deal with, which causes the lengthy project completion time. It's certainly not an ideal model for any project that has pressing time constraints. Frequent risk analysis meetings may alter the whole face of the project by eliminating many of the early pre-defined features. Again this increases the project cancellation risk.

2.2.3 Commercially Off-The-Shelf (COTS)

• Description

COTS is a process model that entails the building of a finished product by integrating "already-made" components for a specific need and use. Here, the applications such as messenger or other "integration-free" applications are not considered; only software that are well contained and may require integration to a proprietary environment (Boehm, et al., 2003) are considered. For example, in a pharmaceutical organization, software called Laboratory Information Systems (LIMS) is COTS. LIMS is software that collects, processes, interprets, reports, and archives analytical laboratory data and results. It can be integrated with a proprietary database, operating system, and other software. A vendor provides LIMS to any pharmaceutical organization that wants to purchase it. LIMS would be integrated and installed in the organization's proprietary environment. The COTS model addresses the integration and installation of software such as LIMS. Here COTS such as LIMS are being addressed.

One may argue that COTS is not a software process model but rather a software management policy due to the fact that COTS constantly changes. This argument is certainly understandable, but going back to the LIMS software example and the software model definition in this paper, the authors support the view of COTS as a model rather than a management policy.

The rising difficulties of the in-house development and the interest of the profit seeking software vendors are the two factors that have given rise to this model. The focus of the model is in the integration and implementation. The vendors provide the ever-increasing variety components to every need of the customers. This gives the software scalability and flexibility, which is difficult for any in-house developed

software to achieve. Also the increased time constraints have driven the developers to an increased interest in COTS model. In reviewing all of the paradigms, the successful key elements are: 1) the user satisfaction, where the user can participate and have a voice in the development direction, 2) the ability to go back and correct things if necessary without any severe penalty or a time delay, 3) scalability and 4) flexibility. Quality is another key element that has other significant consequences such as a legal issue (Cosgrove, 2001). In-house software development is difficult to comply with these. Regardless of any quality measure such as CMM or ISO, it is difficult. The in-house development insufficiencies may lead the team in a frantic mode where it's pressed for time and resources and sometime stifled by office politics, thereby forcing issues where there is no easy solution. COTS is probably one of the solutions here.

- Applications

With today's COTS, almost any type of software is possible. The incredibly deep knowledge and business understanding of customer-centric vendor operations have given the software professionals a freedom to build a product. The COTS model is effective for developing software that is to be used in a non-critical but volatile organizational operation. Volatile, in that it abruptly seeks scalability and flexibility. Non-critical in that the component-based system is off-limits as a mission critical system. In the dynamic business environment, the software for overhead operation or administrative use is a good match with this model. Scalability and flexibility are becoming important aspects in software and systems (Basili and Boehm, 2001). Dynamic business requirements and external business factors demand legacy systems to adopt the volatile environment and the supporting functions and infrastructure to be agile.

From the management's view, it's inconceivable to lose any business operation or output due to a limited system capability. A full operation for the maximum output is what's expected at all times. The "dynamic" business terrain demands "dynamic" navigational tools. A multi-dimensional player who can play many positions and play them at any level on an ad-hoc request is what's needed. Migrating legacy systems is not a frugal choice. This effort is a very resource draining and time delayed intricate process. There is also an internal demand. Getting the humans to adopt and grow with the software requires a gradual and systematic introduction of the layers of software features. Being able to interface with other software and peripherals at various levels is no longer a selling feature but a basic requirement. The advent of XML and other data manipulation tools attest to the significance of software scalability and flexibility.

- "Already-made"

Probably the most attractive point of COTS is that catch slogan. Regardless of what the opposition may say about COTS, the undeniable fact is COTS's complete elimination of design, code, and test of software itself. The well-made healthy features resulted in the increasing dependency on COTS. The vendors' aggressive marketing and customer-centric operation influenced many notable organizations to build their systems with COTS. The bigger incentive of CBS comes in the maintenance stage. Change control management and an ownership of the system for constant software updates and troubleshooting may impede the customers. Answering this dilemma, the vendors provide various service packages that include all necessary steps and remedies. An on-site service man is a common service now.

• Scalability

In retrospect, many legacy systems were scrapped because they lacked the performance and capacity to accommodate the increasing demands of processing and new orders. Although still strong, IBM's mainframe is no longer the main horse carrying the load. The "super" computer is no longer the super computer by today's standard. Increasingly tougher business orders drive the performance expectation so that it is not viewed as just a feature but as a basic necessity. The scalability is the strong point of COTS. Depicted as stacking more blocks on the top of existing blocks, simply adding more to the current performance is what COTS brings.

• Flexibility

Not only is the vertical dimension, the scalability, important to customers, but the horizontal dimension of flexibility is equally as important. This is more challenging as different tasks are to be processed. One real industry example is the chromatography analyzer software. Heavily used in the pharmaceutical industry, the software is capable of not only performing the many hierarchical layers of fine detailed analysis but it's also capable of doing both gas chromatography and high-pressure liquid chromatography. It's unquestionably advantageous to the customers to have a lesser number of systems and yet still have performance capability. COTS is encapsulated and segmented according to its features and performance. With COTS, customers can choose either to go up vertically or horizontally according to their needs.

• Limitations

Currently, there is no industry "accepted" COTS process model (Fox et al., 1997a; Fox et al., 1997b; Hirai, et al., 1998). Obviously even with the "already-made"

components, there needs to be a certain order to build the final product. One can face many pitfalls in just following any conventional process model with no special considerations given to COTS (Hirai, et al., 1998; Brownsword & Place, 1999; Hissam & Plakosh, 1999). Many of COTS limitations are discussed in the COTS section later in this paper. The problems and limitations of CBS are listed as follows (Oberndorf, et al., 1997; Hissam & Plakosh, 1999; Brownsword& Place, 1999):

- COTS marketplace is in constant change
- Constant evaluation of COTS components
- Risk of building mission critical systems with COTS
- Extensive added cycles of prototyping for expected results validation
- Misunderstanding by management on the view of COTS
- Lack of COTS technical information
- Full dependency on vendor
- Uncertain future
- Security Issue

The constant evaluation of COTS for the updates is probably the most challenging step in building a successful CBS and also maintaining one (Reifer, et al., 2003). Evaluation is not a one-time step in the CBS. It's done continuously until system retirement. Many vendors do host a series of user meetings to inform and discuss the customers' feedback in an effort to incorporate the customers' voices in the updates and next releases. The customers tend to upgrade their CBS if a newer version component comes to market. It's hard not to upgrade as the new version component gives a higher

and more efficient system performance. This continuous evaluation not only reflects the upgrade cycles of the COTS vendors, but it also provides a way of identifying new technologies that can support the evolution of CBS. Nevertheless, these user meetings are not all smiles and handshakes. It's a discussion forum where the disgruntled customers bring their complaint lists. The strong influence from the customers in many cases can shape the next release. This exhibits the "survival dependency relationship" between the vendor and the customers. The vendors can't survive without the loyal customers and the customers need CBS for their objectives.

With COTS, many professionals conjure up the difficulties in COTS integration. The integration difficulties embody many of aforementioned bullet items. One restricted area for CBS, although it's lessening, is the mission-critical system development. When it has to be perfect as it can be in all aspects of system, the professionals are still hesitant in CBS. It's the unsure feeling of "I-don't-know-what's-in-this-black-box." The benefits of CBS lead to many other system developments, but not mission-critical development. Adding more to the uncertainness are the lack of COTS technical information, the full dependency on the vendors, the uncertain vendors' future thereby the CBS's future, and the security Issue. In the integration and implementation, all these issues must be satisfactorily resolved.

- Risk

Using 'pre-made' components raises the vendor reliability concern. Choosing COTS, in many cases, would mean a life-long contract with the vendor. The life of the vendor, the product, and the underlying relation with the vendor are the questions to be addressed. Practitioners are voicing a concern for the COTS constant evaluation

(Oberndorf et al., 1997). A continuous version upgrading will give the better technology to the organization but at the same time, it also cost time and resources in the implementation. Data integrity and conversion, compatibility with other existing interfaced software, and retraining are to be addressed. Scalability and expandability, led by proprietary business drivers, sometimes require the CBS to be flexible to meet the demand. The business drive wasn't foreseeable in the project feasibility in many cases.

Early termination of the product by the vendor is another risk. The vendor's sales talk gets the sale but in some cases the vendor would terminate the production and support of the software. This leaves the organization in a dilemma. For the vendor's interest and profit, the vendor would push for the newer software. This leads to the business relationship and trust with the vendors. It's only secure as walking on a thin ice. In COTS, many organizations skip or limit the testing stage. However, some organizations would perform 'white box' testing in firmly assuring the validity of the components in a propriety environment. Smooth and uninterrupted technical support is expected, but not always delivered as the software is well into production.

- Implementation

The expectation from management and users and the underestimation of COTS implementation is an issue that needs to be addressed. In some cases, COTS may be more to deal with than traditional in-house software development (Oberndorf et al., 1997). Interfacing with associated software is the difficult part. Typically many organizations deploy various different COTS from different vendors. One can have a proprietary software, a brand X database, a brand Y operating system, a brand Z server, and so on. All these must be synchronized to perform harmoniously toward one goal. A

feasibility study before the implementation is critical as is the role the vendors play during the implementation stage.

- Impact on customers

A case where a CBS is simply replacing a manual process is where the most COTS benefit is shown. While least disturbing the current business process, most end-users are ready to use the software with minimal training. Being able to shop for the desired components according to the current situation in hand is another benefit. Being able to select the components based on their strengths and weaknesses can limit the shortcomings of CBS (Balk and Kedia, 2000). "Defects or design issues can be rapidly worked around by adding low level coding to the component interfaces." (Balk and Kedia, 2000).

Probably the biggest advantage of COTS is the "no testing required." While some may argue this point (Oberndorf et al., 1997) for the interface testing, generally COTS itself doesn't require any functional testing. Economically, using COTS is one of the cheapest ways of getting software. COTS costs only a fraction of the multi-million dollars that an organization would spend for in-house software. On the other side of the coin, there are some negative impacts. There is a growing voice that "COTS is more work, not less work" (Oberndorf et al., 1997). Relative to in-house software, activities such as integration, implementation, evaluation, updates, and customization is increased, causing the customer to be more pro-active. Maintaining a healthy professional relationship with the vendors is another concern. As COTS entails absolute vendor dependency, a long-term trusting relationship must be maintained from both the vendors and the customers.

- Web-based COTS Integration

COTS continues to draw a growing number of followers. With the traditional software development challenged by agile software development, the ever-increasing complexity of the software development practice has some organizations withdrawing their resources from in-house development practice. The other route is COTS. With a concentrated effort in integration and implementation, COTS eliminates the aches and pains of designing, coding, and testing. With the rapidly growing spectrum of vendor services, COTS is a tempting and viable option. Some research items are: 1) proposal of a model for web-based COTS development focusing in integration and implementation, and 2) measurement and assessment of the empirical results between CBS and non-CBS in equal environment.

2.2.4 Prototyping-Based

- Description

Prototyping is probably the most widely used and accepted software development practice. Prototyping has become the tool in requirements engineering (Dorfman, 1997; Gomaa, 1997; Floyd, 1984; Rettig, 1994; Lichter, et al., 1994). Two things that conjure up interest are the quickness of prototyping and the user involvement. Prototyping is defined as an approach based on an evolutionary view of software development, affecting the development process as a whole (Lichter et al., 1994).

With the prototyping method, users experience more "control" in the project than with other models. The birth of prototyping is largely a response to the waterfall model limitations, slowness and no user involvement. The 'visualization' of the long and thick requirement document frustrates the users. This 'visualizing' is where the benefit of

prototype model comes into play. The "hands-on" experience displays its uncanny ability in defining and capturing the specifics and details of requirement. This is because most users do not know exactly what they want and how they want it (Gomaa, 1997).

User involvement is such an inextricable activity of software development that even the waterfall model recommends it (Royce, 1987). With the success and high user satisfaction from prototyping, many developers gradually adopted the prototyping process model in developing the entire software. The concept is equally effective as a software process model as in the evolutionary model and the incremental model (Sommerville, 2000). The evolutionary model is an incremental approach process that incorporates each increment in every increment cycle. After each increment, it gets to be put into use in production. Each earlier increment helps to define the requirements of subsequent increments. The increments are done serially. Within each increment cycle, the stages are similar to the linear sequential stages.

The incremental model is a model where the overall architecture is developed first and then each feature has its own cycle where it gets to be incorporated at the end of the cycle. A good example is the cleanroom model (Prowell et al., 1999), also known as the cleanroom IBM model. It's an excellent quality model for reducing errors. When all increment cycles are incorporated, the finished software is put into use in production.

- Applications

Due to its superlative effectiveness, it is widely used. There are a number of various prototyping-based paradigms. Adhering to the scope of this paper, they are not discussed in any detail.

Under the application, the two prototyping-based models are in the same category as the spiral model. An ideal project is a large project that is the first of its kind, unproven and risky. Quality-focused project is applicable as well. Each increment cycle eliminates errors, and undesirables.

• Limitations

Ironic as it is, the reason for most prototyping project failures has to do with lack of end user involvement (Lichter et al., 1994). 'How much' and 'what' are the questions in dealing with the user involvement. Also, there is no explicit guideline in getting the most out of the user involvement. The limitations can be listed (Sommerville, 2000): 1) prototyping doesn't necessarily reduce the software development time, 2) sizing up the initial increment and subsequent increment is very difficult, 3) the "endless" repeated cycles of prototyping can get out of control, and 4) estimating the total project cost and resources are difficult. The most stifling thing is the appropriate increment size and level. A system must have minimal functions of all parts in order to be used. Questions arise such as what gets included in initial increments and to what extent will they be included. All these are serious questions to deal with.

2.3 The Contemporary Paradigms

From the early days of the software crisis, the importance of having a set of orders in the development was recognized and respected. The increasing size and complexity of developing an application sought for a structured framework that would lead to a more organized and better-managed development practice (Royce, 1987). The linear

sequential or waterfall model, along with its variations, and the famous prototype-based models enter the scene and they appeared to manage the demand.

The advent of Internet and e-commerce has added yet another and probably the most challenging task to the software engineering community (Baskerville, et al., 1992; Cusumano and Yoffice, 1999; Baskerville and Pries-Heje, 2001a). The outside influence, mainly the business ecosystem, has always been the leading and primary driver in the trend of software development processes. One underlying technical factor that allows e-commerce to enjoy its exponential growth is the direct link connection between the primary supplier and the main consumer (Stalk and Hout, 1990). This interprets to a much wider and diverse user base, a more effective and faster application, and a higher demand for a newer and better version (Goldman, et al., 1993). The responsibility of meeting and accommodating these issues would belong to many stakeholders but the technical aspect belongs to the software engineers.

Here one sees a similar phenomenon comparing back to the early days of the software crisis; a new order is required in meeting given business demands (Baskerville and Pries-Heje, 2001a). The need and place for the established software development process models such as linear sequential or incremental models are clearly understood. What is being discussing here is one of many possible approaches in dealing with the growing demands of small or mid-size (internet-related) applications (Baskerville and Heje, 2001b; Cusumano and Yoffice, 1999). The legacy systems or large transaction Management Information Systems would be developed with more established or structured software development process models.

Application type

- Small or mid-size (Internet-related) application

Two areas where established models are lacking;

- unpredictable market volatility and
- constantly changing user requirements

Having said these, an attempt is made in listing some of limitations and shortcomings of the established software development process models

Table 2.3 Factors Displaying the Shortcomings of Established Models

Factor	Shortcoming
Speed	Requires a much shorter development time in meeting the market time
User requirements	Dealing with constant changes in user requirements
User participation	User participation is never enough
Bureaucracy	Agility in quick response to outside influences and factors
Frequent version release	Meeting the market (customer) demands and volatility

The table is not complete by any means, but it does share some of the main points.

- Speed – It may be the biggest reason why linear sequential model or other established models cannot be an answer to today's dilemma in developing small to mid-size internet-related applications. The underlying fact is the fierce market

competition, "time-to-market." With the technical parity among organizations, today's Internet business battles are heavily influenced by "who gets there first" (Goldman, et al., 1995; Stalk and Hout, 1990) The newer business model dealing with "time-to-market" demands or requires the same quickness from the required software development and the established models can't do this.

- User Requirements – When the business market was predictable and conforming to a forecast, the user requirements were pre-defined and well contained. However, the ever-complex business terrain and the advent of e-commerce have completely changed the way the user requirements document is being prepared. In small or mid-size Internet application development, the user requirements don't stop but continue until the last minute before the application release. This behavior can be contributed to the users themselves but may largely be due to the market volatility and growing demands of the customers. A last-minute new user requirement should not be considered as an abbreviation but a norm in this context.
- User Participation – The level of user participation in the software development process has been increasing as more and newer software process models were available, but the persistent users requirements problems desired more user participation. The established models don't allow this to happen. The closest practice is the prototype-based model such as the incremental model where users are to review during scheduled review periods. This appears to not be enough in the context of small or mid-size Internet application development.
- Bureaucracy – In the established models, many interest groups or their representatives across organizations are present in the development team. This

leads to a well represented but large team that may be slow to respond to user requirements and changing market demands.

- Frequent version release – As a common characteristic in small applications, frequent version release satisfies the users and market changes. But using more established models does not give this flexibility.

Again, the established models are still the main models and most mission critical systems are being developing using these established models. But now a new breed of models as a response to the changing business market has arrived.

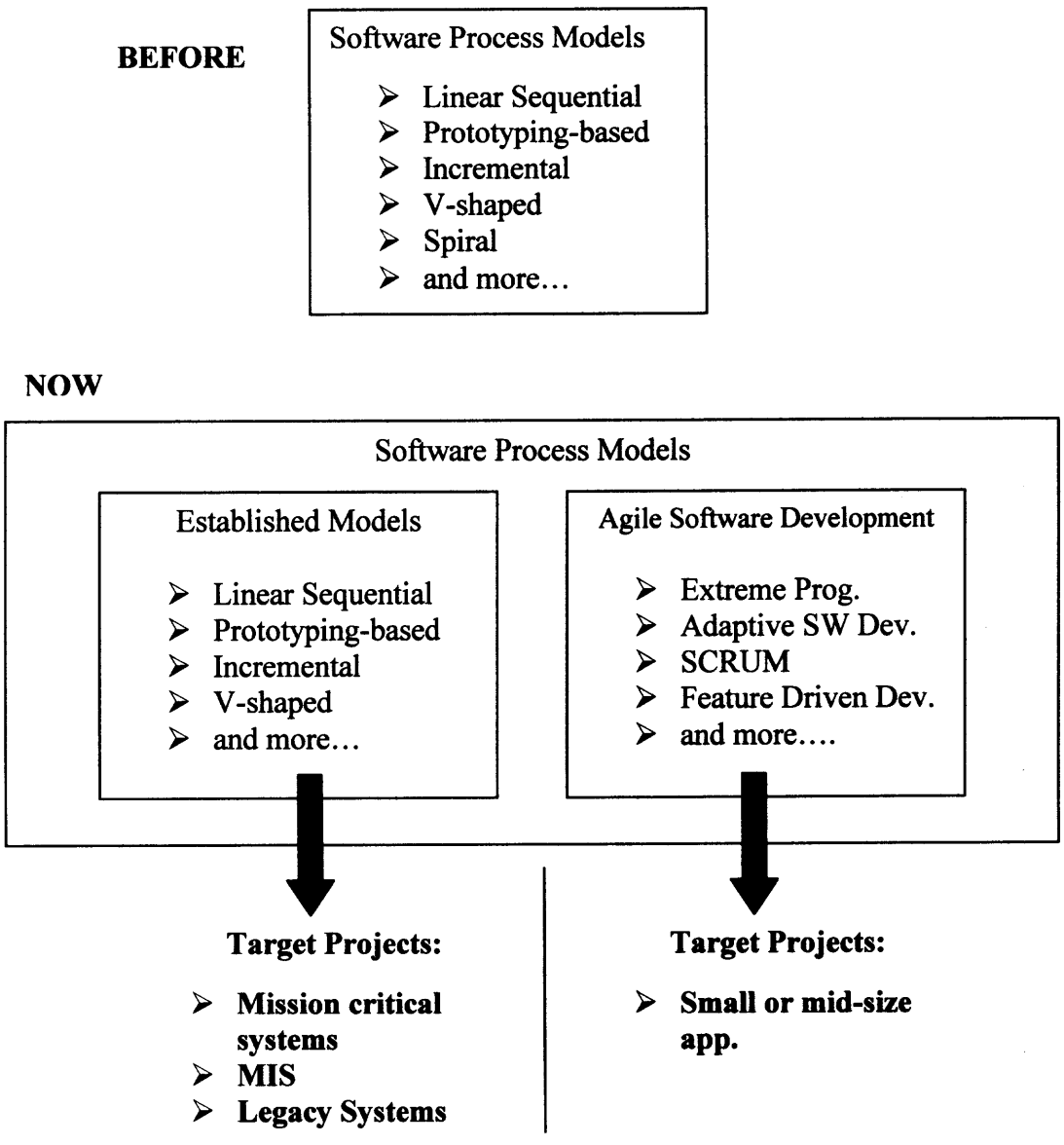


Figure 2.1 Agile software development in software process paradigms.

It would be almost impossible to use a model for another target project. For example, one cannot and maybe should not use XP in developing a mission critical system or military system. On the same token, one would be wrong to use linear

sequential model in developing a small internet-related application that embodies constantly changing user requirements and a critical "time-to-market" factor.

2.3.1 Agile Software Development Introduction

While it is also called "light" methodology or "eMethodology" (Baskerville & Pries-Heje, 2001b; Nawrocki, et al., 2001), "agile" software development is the more popular term (Cockburn, 2001). According to the Manifesto for Agile Software Development (Agilemanifesto website), the shared values are:

- *Individuals and interactions* over processes and tools
- *Working software* over comprehensive documentation
- *Customer collaboration* over contract negotiation
- *Responding to change* over following a plan

One can easily see that the values are completely opposite to the traditional software development paradigms such as Linear Sequential. There is no "locked" commitment (Abrahamsson, et al., 2003). Agile Software Development can be defined as, by this paper, a collection of the unorthodox software development paradigms, characterized as adaptive, result-oriented, and human-centered, that is extremely focused in completing a premium quality software in the shortest time possible (Williams and Cockburn, 2003). A more practical description is:

- Requirements are a "moving target"
- Customer satisfaction is my first goal, my second goal and also my last goal
- If my customer says "Jump!", I will ask "How high?"
- If I don't adapt, I will die the next day

The time is the key requirement as its primary objective is completing a job in the shortest time possible. In achieving this objective, it eliminates any "time-consuming" and "long-term value" activity such as generating a bulky, "hardly-used" software document (Abrahamsson, et al., 2003). To illustrate the contrast between the traditional and agile software paradigms, the following comparison between Linear Sequential and XP is made.

Table 2.4 Comparison Between XP and Linear Sequential

XP	*Linear Sequential*
Informal & very direct	Constraints
Changing requirements	Fixed requirements
No or min. documentation	Requirements Eng.
FtF (anytime)	Comprehensive documentation
Frequent releases	Scheduled meetings
Self-empowered	One final release
Self-contained	High dependency

The general perspective is that the traditional paradigms are structured, rigid, and slow and the agile paradigms are unstructured, flexible, and fast. All other agile paradigms share most of XP's characteristics. Probably the most profound item is the "changing requirement." This really sets the tone on how the development will be conducted: completely user-driven, user-focused, and for user's satisfaction (McCauley, 2001; Cockburn and Highsmith, 2001). How each paradigm brings the "shared value" is carefully examined.

2.3.2 Extreme Programming (XP)

Extreme programming introduces a new paradigm that is off-the-wall, but yet is a pragmatic approach (Schuh, 2001; Nawrocki, et al., 2002; Kircher, et al., 2001; Cohn and Paul, 2001). Pair programming, no documentation, no formal requirement engineering, no up-front design, and other anecdotal features manage to produce high quality, frequent working releases, and high customer satisfaction.

Born in the mid 90's by a group led by Kent Beck (Beck, 2000), today XP is becoming the debate in software engineering (Choi and Deek, 2002; Keefer, 2002). According to XP experts, XP is perfect in "risky projects with dynamic requirements." Extreme programming has eliminated the software process stages that are, in XP's view, considered unnecessary. The formal requirement engineering is replaced by a few index cards with user stories written behind the cards in plain English (Nawrocki, et al., 2002). There is a designing stage that generates a stack of binders, and documentation is optional. There is no "forward" coding allowed. In other words, only code what is asked for at that time. Instead of the traditional one-man programming, pair programming is exercised. Code review and unit testing are constantly done between the pair programmers. A frequent release cycle is observed as working software is always in production for feedback for the next release. A customer is on-site answering the developers' questions and providing the subtle details of requirements. An acceptance test by the customer confirms the requirements and approves the release. Following is a simple pictorial XP workflow:

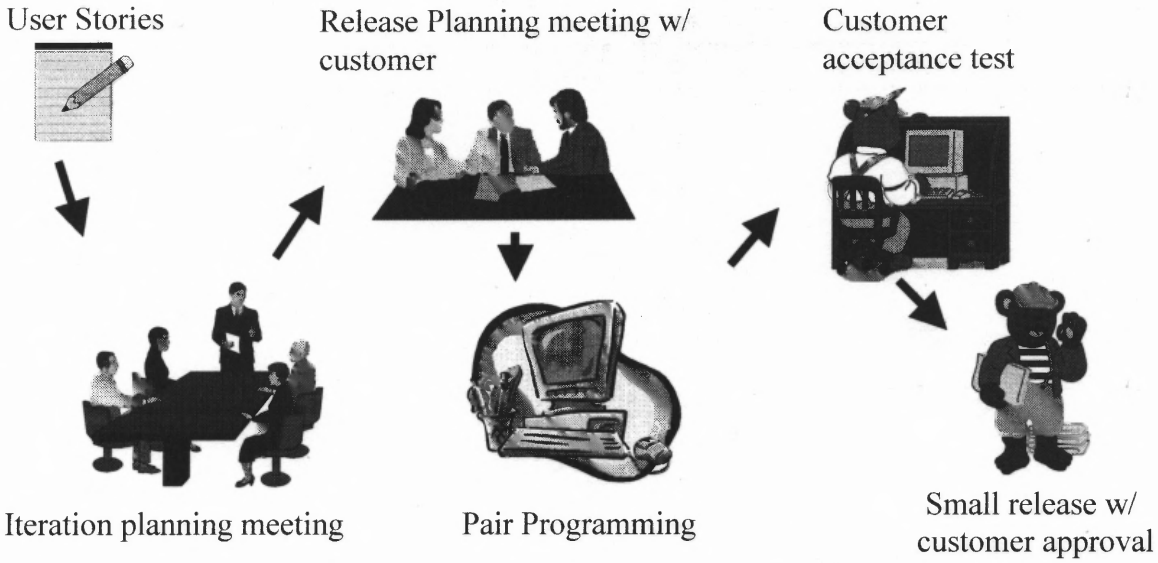


Figure 2.2 XP workflow. This depicts the workflow of XP.

Table 2.5 XP Work Flow Terms Description

Stage	*Description*
User Stories	This is a description or a story describing what's to be done. Written by the users and in plain English
Iteration planning meeting	A meeting where the on-site customer debriefs the group of XP programmers on the index cards of user stories
Release Planning meeting w/ customer	The on-site customer listens to the user stories priority, risk ranking, and other feedback from the XP programmers
Pair Programming	All work is done by pairs of programmers. Design, code, code review, debugging done by the pairs.
Customer acceptance test	On-site customer tests the batches of completed work
Small release w/ customer approval	Upon the acceptance, the work is released.

Extreme Programming not only yields high satisfaction from the programmers but from the managers as well (Grenning, 2001). This is largely due to the frequent releases and to the fact that a manager can actually "hold" a piece of a completed project. The programmers love XP, as it has freed them from tedious documentation and other relevant chores (McCauley, 2001). Extreme programming stands on twelve core practices. Here are the complete twelve core practices of XP:

Table 2.6 XP Twelve Core Practices

Practice	*Description*
The planning game	Customers define application features with user stories.
Continuous testing	Designers write automated unit tests up front and run them throughout the project.
On-site customer	A customer representative remains on-site throughout the development project.
Small releases	XP teams integrate code and release it to a repository every few hours and in no case hold on to it for longer than a day.
Refactoring	XP teams frequently revise and edit the overall code design, a process called refactoring.
40-Hour Work Week	Programmers work only 40 hours per week; there's no overtime.
System metaphor	XP teams use a common system of names and descriptions.
Pair programming	Programmers working side by side in pairs, continually seeing and discussing each other's code.
Continuous integration	XP teams put small code releases into production early.
Simple design	Teams emphasize simply written, object-oriented code that meets requirements.
Collective code ownership	All programmers have collective ownership of the code and the ability to change it.
Coding standards	Programmers must follow a common coding standard so all the code in the system looks as if it was written by a single individual.

These twelve core practices yield the full benefit of XP only when they are used together. However, as dramatic as these practices are, many are only in partial use (Grenning, 2001). But such partial use may be a good introduction to an "XP-unbelieving" organization. Extreme programming is mainly geared to the small to mid-size application. Thus, the drastic practices are not so drastic to such application development, especially for applications with a small number of users.

• Pair-Programming Introduction

In speaking of XP, a discussion of PP is inevitable. Pair programming is defined as "a programming activity where two individuals sit next to each other sharing one keyboard and one mouse." The high level of satisfaction is a known fact of XP as a study showed a group of programmers overwhelmingly preferred PP over one-man programming (Williams, 2000). Given the fact of two heads programming with just one keyboard and monitor, which is still the fact that is difficult for some to accept, the psychosocial probing deserves some consideration.

The "C3" team reported the first successful large-scale empirical work of PP that was done in industry (Anderson, et al., 1998). It's a showcase of the strengths of XP and PP. Under the real situation, the C3 team put the runaway project on the right track by completing the project on time. For the empirical work, a 45-minute long "lab-controlled" experiment (Nosek, 1998) was conducted where the programming professionals were divided into either PP groups or one-man programming groups. The "quick" experiment revealed that in comparison to the one-man programming groups, the paired groups 1) generated a longer completion time, 2) generated a higher quality work, 3) felt a higher enjoyment and confidence in the work, and 4) saw that the more

experienced programmers performed better. The experiment was conducted with only 15 subjects.

A few PP experiments in university settings using student subjects were performed. A small scale (only one team, 4 subjects in the team) yearlong student XP experiment reported a negative PP experience (Kivi, et al., 2000). Four months into the experiment, PP was dropped by the subjects because "team members felt that paired programming was a waste of time." The authors attributed the results to the subjects' lack of prior software development experience and the difficulty of finding common hours to pair-up. The subjects were given the freedom to pair-up at their discretion.

A more controlled experiment, which was done, consisted of 41 student subjects (Williams & Upchurch, 2001). Initially, "jelling" sessions were held to allow each partner to "settle in." The experiment also looked at what impacts the grade point average, gender, and personality types could have on the outcome. In a comparison to the one-man team, the results revealed: 1) a slightly longer (15%) completion time, 2) a slightly lower (15%) defects rate, 3) a very high satisfaction (96%) from the subjects, and 4) no "conclusive findings" on the possible impacts of grade point average, gender and personality types. However, the experiment only had seven male-male teams, four male-female teams, and one female-female team. Contrary to the common expectation of "get the job done in the half of the time," most of the empirical works showed a slightly longer completion time than one-man programming. Nonetheless, the PP group did exhibit quality work (much lower error rate) and overwhelming programmer satisfaction and confidence, which is significant factors in sustaining a long successful project.

• Extreme programming, too extreme?

To complete the overall picture of XP, it would be an injustice not to discuss XP's limitations. Recently, a growing number of XP skeptics have voiced their concerns in both literatures and Internet newsgroups (Choi and Deek, 2002; Keefer, 2002; Stephens, 2001; Saiedian, 2003). Any of the active XP debates are easy to spot on the web at anytime (XP newsgroup; XP website). Aside from some ignorant opposing comments, there are some earnest sentiments and legitimate concerns. In an effort to better understand the opposition's validity and XP, some of them are presented in following.

• Opposing views to XP

The sources of the opposition views are gathered through a search in literatures, Internet newsgroups, and other online discussion forums. As is with the nature of Internet newsgroups and online discussion forums, many postings were authored either anonymously or in pseudonames. However, there were many postings with valid personal names and professional affiliations, mostly from the XP authors and XPers. The posts from mostly the XP opposition or anti-XP posts have been carefully evaluated and analyzed.

The list is a categorized collection of the most common complaints that have been gathered from the sources. Because of the vast amount of complaint posts and their variety, the list is only a portion of them. The profile of a typical online discussion participant is an experienced professional programmer who either did try XP or encountered XP indirectly through published literatures or in a managerial capacity. It was apparent that the legitimate posts were likely from the professionals who have

actually have tried XP, and this was found to be very important in understanding XP. The following paragraphs are described per opposition views.

"Extreme programming is simply not a good method in producing high-reliability software." (Stephens, 2001; Keefer, 2002)

A quote from the article (Keefer, 2002) says "*however, in order to successfully develop software we have to limit the degrees of freedom our nice languages, tools, and personalities offer and select from all the practices and techniques available, and formal quality models are useful guides.*" Everything about XP is at issue. Extreme programming itself goes against all good software practices. The C3 project (Anderson, et al., 1998), which is the showcase XP example, is misleading because the project was cancelled after being nearly four years in XP mode (Keefer, 2002). XP, the "humanistic method," (Martin, 2000) is risky and unaccountable. It is true that the human factor is the key to success but by the same token, it can also be the key failure factor. Decades of hard work and caring minds have established many prudent software process principles and there is definitely a set of irreplaceable values, which cannot be undone by XP.

"Extreme programming is not good in systems that present many constraints and strong security requirements." (Keefer, 2002; Online message forum on XP website)

Extreme programming's concept is "do it as simply as possible and do it fast." Extreme programming is not a good match for a project that carries a bag of many interacting constraints of other associated entities. If a project, such as a military missile guiding system or a nuclear reactor controller system, where the security and specified constraint requirements are of more value than the project, then XP is definitely not the choice. Government regulated industries do not use XP at all as the features such as no

documentation and no up front design are completely unacceptable per government inspection. This tells a lot about XP as a prudent software process method.

"On-site customer will not work." (Stephens, 2001; Online message forum on XP website; Yahoo XP website A; Yahoo XP website B)

As the first on-site customer of the first XP project, C3 (Anderson, et al., 1998) was burned out after only a few weeks into the project and was not adequately replaced (Keefer, 2002). To relieve the load, a pair customer was tried, but the pair customers would not help either, as they don't speak in "one voice" (Online message forum on XP webiste). Human memory is bad and should not be used as an accurate account. The requirements of software must be accurate and specific. On a more practical level, the on-site customer would likely be a junior in his organization, which means his answers to the programmers' questions may not be correct. No company would spare a senior staff to a project that requires on-site for months or even more than a year. Furthermore, who would take the job if the job requires nothing but answering random questions of programmers whole day? (Stephens, 2001).

"No documentation is insane." (Stephens, 2001; XP newsgroup; Keefer, 2002; Online message forum on XP website; Yahoo XP website A; Yahoo XP website B)

XPers insist that the code itself is a document. This is an oxymoron. The documents are mainly there to explain what all those thousands of lines code mean. "Considering maintenance and usage aspects, proposing 'no documentation' is clearly a professional malpractice (Keefer, 2002)." Although exercising good programming practices such as inserting full conforming code comments is helpful, it's still not a substitute for the documentation.

According to XP, a large stack of binders of documentation is no short-term value (Beck, 2000). This is completely incorrect. Software documents are not only for the long-term value such as production maintenance, but the document itself is a part of the software and is inseparable. How about the change control documentation? What got fixed, what's outstanding, what's on standby, and more. There is a definite short-term value. Exercising good programming practices such as inserting a long code comment, a description in the beginning, appropriate readable code comments for each significant piece of code, and indentations and "white spaces" for easy readability are what every programmer must adhere to regardless of documentation or no-documentation.

In government-regulated industries, software is not complete software without its complete set of documentation. After raising this no-documentation concern, XPers have lowered their point to "optional or minimum documentation," but the true spirit of XP is no documentation. Also, there is no such thing as optional or minimum documentation. It's either you have it or not. Here, Extreme programming directly collides with the traditional software process value.

"No up-front design will only frustrate the programmers if not give up." (Stephens, 2001; XP newsgroup; Keefer, 2002, Online message forum on XP website)

Just as with the absence of documentation, XPers do not exercise any formal design stage either. After collecting the index cards of user stories, immediately they start to code. A structured and careful design is again eliminated in favor of the spontaneous pair collaboration, integration, and programming. Again, XPers explain that the design is in the code itself. For an outsider, this is a hard one to accept in programming. Programming without any up-front design has been epitomized as programming

ignorance. To many, this would be like a platoon going to a battle without any plan and plan as they battle, who would survive?

"Extreme programming's pair programming is a double-edged sword." (Stephens, 2001; Kivi, et al., 2000; Muller and Tichy, 2001; Yahoo XP website A; Yahoo XP website B)

Pro XP articles are rushing to share the "good news" about XP, how great the pair programming is and its wholesome results (Yahoo XP website A; Yahoo XP website B). Is this always expected? The psychological aspect of pairing up with another individual warrants a careful preparation. It is true that a good pair can bring the synergy (Barnes, 2001; Haungs, 2001) that XPers are boasting about, but on the other side of the coin it can also bring a disaster. Possible reasons for a failure can be in differences of personalities, work ethic, job satisfaction, personal agenda, the relationship with the paired-partner, cognitive approach in problem solving, and some unfortunate social issues such as political and power management.

In programming, "*it takes a certain kind of programmer to want to work in harness with another. Many prefer to work for bursts of creative time in isolation...* (Glass, 2001)." It's tough when one is trying to think and other is trying to talk. The protocol of pair programming is vague as well because there is none, at least not an official one yet. Some articles do suggest some general guidelines (Williams and Kessler, 2000), but enforcement is unlikely.

Many who are planning to try or have tried XP are mostly concerned about this pair programming (Kivi, et al., 2000; Muller and Tichy, 2001; Yahoo XP website A; Yahoo XP website B). As human nature has it and with the "Internet time" as the

primary driver for the business ecosystem, the time factor eventually forces the more experienced programmer to drive the keyboard as the less experienced programmer would "assist" in getting the job done quickly. This is what's happening to a typical pair that is told to exercise XP with a project deadline hanging over their shoulders (XP newsgroup; Yahoo XP website A; Yahoo XP website B).

• Discussion on the views

"Extreme programming is simply not a good method in producing high-reliability software."

It's unfair to say that software developed by XP is not highly reliable software in general. The aggressive unit testing practices of XP have been praised even by the skeptics and some opponents (Glass, 2001; Saiedian, 2003; Yahoo XP website A; Yahoo XP website B). Anecdotal as it is, the XP results (Williams, et al., 2000) are surprisingly high in quality.

Despite the fact that the XP process is very questionable, the results are promising, hence this XP euphoria. There is an effort (Nawrocki, et al., 2001) to bring this strange and extreme method more into the software process practice mainstream. To discuss the reliability issue in detail would be beyond the scope of the paper. However, the issue may be accepted accordingly to one's background. In the software community, the programmers tend to favor the code-centric XP, hence they do not agree with the reliability issue. On the other hand, software inspectors and engineers completely agree with regards to the reliability problem (McCormick, 2001).

"On-site customer will not work."

Here, the opponents' view is well founded and this is echoed repeatedly in a number of posts (XP newsgroup; Online message forum on XP website; Yahoo XP website A; Yahoo XP website B). In many real life cases, it would be lucky to have any willing customer in a software development process. In the traditional way, only the requirement engineering has the customer's involvement, but in XP the customer is on-site and in the process from the very beginning to the actual delivery. This creates a very stressful situation for the on-site customer. The on-site customer must make himself available to the programmers' questions at anytime. Worse yet, this is all he does the whole day. If this was known, undoubtedly no senior staff would want it.

This is another reason why XP must be used in a small project or else the poor on-site customer will experience burnout just like the first on-site customer in the first XP project, C3, did. Use it in a small software development project with a short life cycle so that the on-site customer's job is shorter and less susceptible to burnout. Also, this is a good example of how important the requirement engineering is for the success of a project. The on-site customer is the most extreme case of requirement engineering and it will not work. There is a suggested alternative (Keefer, 2002), but this will go as the field of requirement engineering advances.

"No documentation is insane."

After massive opposition to this, XP lowered its standards to "optional or minimum" or "as you wish" documentation. There are probably many reasons why the documentation should exists, but XP's view to "only the long-term benefit" (Beck, 2000) documentation is a lesser value than the code, the end product and delivering the

software on time. This view can be discussed, challenged, and rebutted endlessly between the XPers and their opponents. The continuous pair review, pair integration, unit testing, and the final customer acceptance test do not and cannot fill the void left by the missing documents.

Coding is definitely not an isolated activity. All associated activities are involved and interact with the coding activity and documentation is required in defining and structuring this. The documentation has always been the map in telling how much progress is made and where to go from here. "Document appropriately" but keep the XP pace is what the debate is.

"No up-front design will only frustrate the programmers if not give up."

This really challenges the way most programmers are think and approach before coding (Fowler, 2001). Without having any kind of architecture up front, it forces the programmers to "design as you code." This really turns off many experienced programmers (Glass, 2001; XP newsgroup; Keefer, 2002; Yahoo XP website A; Yahoo XP website B). One alternative is to provide a written minimal but concise high-level architecture that would lessen the frustration and then the programmers may "design as they code" for the lower-level architecture. To many of these questions, the XPers' attitude of "practice, practice, and more practice" certainly does not help to ease the issue.

"Extreme programming's pair programming is a double-edged sword."

Besides all of the eyebrow raising features of XP, the pair programming is the most intriguing part. Ever since man created the computer, programming was and still is a

one-man activity. It's one's world where he let go of his creative mind. Enter XP, and now sharing is the key word. As XP itself is extreme, this certainly bills to it.

While pair programming yields the benefits of knowledge transfer, pair pressure, and confidence building, pair programming can also introduce bad programming habits under the name of "do it this way, it's much easier." The senior programmer may influence the junior programmer on how to "cut corners" against the standards. With the project completion due date hanging over their shoulders, this is a likely scenario. Also in (Williams and Kessler, 2000), the authors make a point of the partner catching "the most glaring errors, those errors that anyone else can see in an instant" (Weinberg, 1998) but that the other partner just can't see. What if both partners can't? The pair has been "synchronized" in their programming. They have worked long hours, maybe days, together and maybe they both just can't see the glaring errors?

Despite some skepticism (Glass, 2001; Kivi, et al., 2000; Mueller and Tichy, 2001) and abhorrence (Stephens, 2001), some accept it well (Barnes, 2001) and the empirical studies are promising (Anderson, et al., 1998; Williams and Kessler, 2000). A quote from (Barnes, 2001) states "*Sometimes if you're coding alone, you end up going off on the wrong thing for a while, If you're 'pair programming,' that doesn't happen, or it doesn't happen for very long...As soon as one person runs out of ideas, the other person just picks up on them.*" The general idea of pair programming is catching on, but practitioners are seeking more guidelines in pair programming (Kivi, et al., 2000; Muller and Tichy, 2001; Yahoo XP website A; Yahoo XP website B).

Currently, there is no explicit guideline or set protocol for pair programming. A quote from (Muller and Tichy, 2001) states "pair programming is adopted easily and an enjoyable way to code. However, it is unclear what type of work not to do in pairs and

how best to structure pair interaction." Simply pairing two programmers and expecting the result is certainly not achieving. In a pair, programming, but also review, integration, design, and testing occur. This adds more to the pairing process and protocol.

• Summary

Extreme programming's principles, benefits, and lastly the opposition view have been reviewed. XP, the humanistic approach and code-centric method, is viewed by many to be in an intermediary state (Beck, 2000; Riehle, 2001). Extreme programming is not complete; it's still in its infancy. Hence a large opposition body exists. Nevertheless, it's refreshing and exciting for the agile software methodologies (Cockburn, 2001), which will be significant in this "Internet time." XP's ultimate goal of "do little as possible for only the required" has thrown out many of what have been the essentials of good software process practices. Extreme programming is a pro-human method that strongly believes in the human ability as the ultimate key to success (Martin, 2000). The ability to make sound judgments, "use appropriately," and keenly focus on the goals are all human factors.

It appears that many who have tried XP voice that the initial implementation was very difficult. Some of XP's features were adopted and some were not, but in the end all have shown a positive feedback on the XP experience (Grenning, 2001; Haungs, 2001; Karlsson, et al., 2000; Kivi, et al., 2000). The users are already customizing accordingly and adopting selectively from XP's features (Karlsson, et al., 2000). Despite the "extremes," XP appears to be finding a niche in the ever rapidly evolving software process field. A quote from (McCormick, 2001) states "*a one-size-fits-all development*

process does not exist. Software projects vary wildly in technology, size, complexity, risk, criticality, regulatory, and cultural constraints, and many other key variables. .... there is a sweet spot where XP will flourish..." There is no right or wrong in XP today. These debates continuously reveal the nuggets of XP and the "contribution" of XP shall benefit all.

2.3.3 Feature Driven Development (FDD)

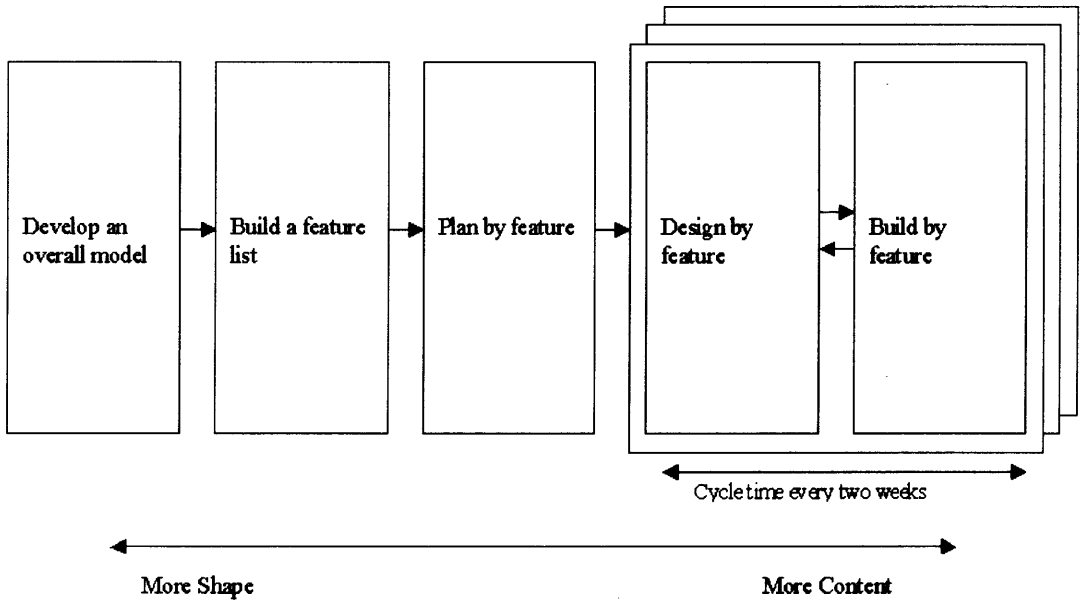


Figure 2.3 Feature Driven Development (FDD) workflow.

Just as with the other agile models, Feature Driven Development (FDD) preaches good talents and the elimination of the complicated process execution. The efficiency comes from the feature-grouping practice. FDD largely can be divided into four stages: 1) Develop an Overall Model, 2) Build a Feature List, 3) Plan By Feature, and 4) Design By Feature and Build By Feature (Palmer and Felsing, 2001). The leader, also called the chief architect, leads the developing team in cognitive walkthroughs in the creation of first draft overall software architecture. This is done similar to XP with user stories, but FDD also accepts written specification documents. The repeated walkthroughs bring many layers of architectures, from the lower level to the highest level. This gives the team a "map."

A feature is defined as "a small piece of client-valued function expressed in the form <action> the <result> <by|for|of|to> a(n) <object>". However, it is arguable as to what constitutes a "small piece." One can have an action and in that action there may be many associated sub-action items. The important thing to remember is that the features are the minimum features that are "needed for the system to be of value to the business." This is similar to XP's "do only what you need for now." Plan by feature is where teams of programmers are assigned to the defined feature groups. Usually, the programmers are assigned to two or more teams. In object-oriented programming languages, like Java, teams are also assigned to own particular classes identified in the overall object model.

The FDD efficiency shines in the last stage of design by feature and build by stage. Here the actual releases come out every few weeks just like in the XP process. The coding happens and the delivery of a working product occurs every few weeks.

Another benefit of FDD is the "Track by Feature" concept. It allows the team to track and report the progress with an exact percentage. For example, a team decides on the percentage distribution as follows:

Domain Walkthrough	Design	Design Inspection	Code	Code Inspection	Promote to build
1%	40%	3%	45%	10%	1%

After each completion of a feature, the percentage is filled in using color-coded marks (green for 'complete,' blue for 'in progress,' and red for 'requiring attention'). This large, wall-posted visual progress chart allows the chief programmer to plan and schedule on a forward basis. Like other agile paradigms, FDD uses strong user involvement, iterative process, and constantly changing requirements. A salient point is

that it possesses a keen management plan, the 'Track by feature.' It's very detailed and specific. Being able to perform with other agile paradigms is another strong point of FDD. Largely for small to mid-size application development, FDD provides an excellent project development framework and accurate development progress reports.

2.3.4 Adaptive Software Development (ASD)

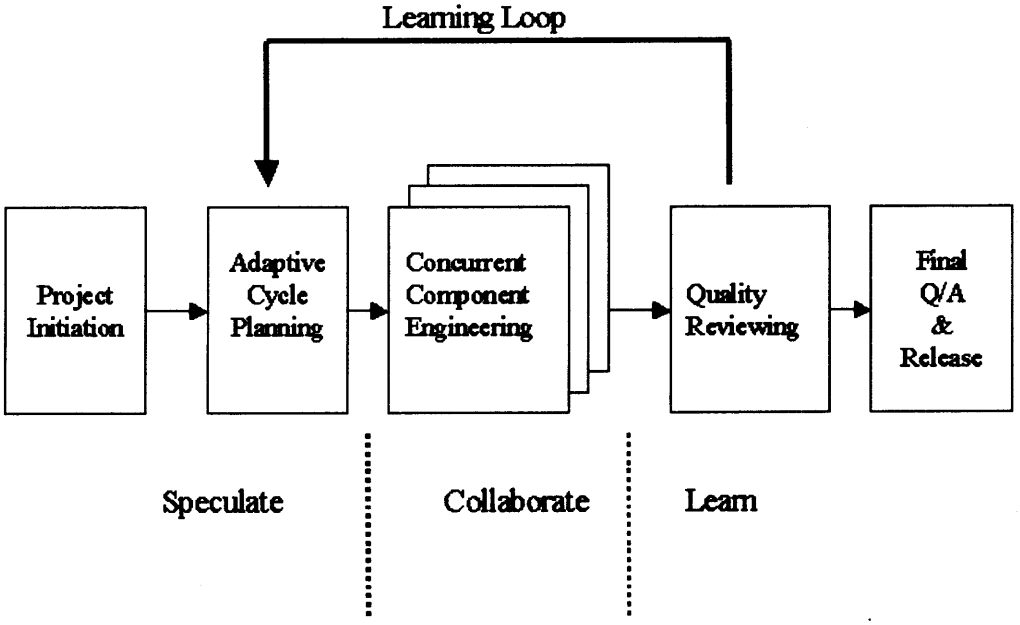


Figure 2.4 ASD workflow.

Adaptive Software Develpment is a diametric approach, believing in adaptation rather than optimization (Highsmith, 2000). As the above figure depicts, it's a cycle process of continuous collaboration, learning, and speculation. First, the development team speculates on what is to be done. This process also takes into consideration the lessons learned from earlier projects. Second, the distributed development team collaborates on the items that are suggested from the speculation stage. This is where the iterative cycles happen. Building by each component, it utilizes and adapts to any

feasible tools in meeting the goal. Finally the team would learn on the release for the next release. In that "learning loop," the lessons that are learned from a particular project are archived for use in the next project's" adaptive cycle planning."

This paradigm is not confined to any particular pre-defined technique, rather it exercises laissez-faire, an "if it does the job then use it" approach. Theoretically, ASD is born out of the Complex Adaptive Systems theory (CAS) (Holland, 1995; Dooley, 1996), where the three key principles are: 1) order is emergent as opposed to predetermined, 2) the system's history is irreversible, and 3) the system's future is often unpredictable. An accurate account of ASD's imposition (Riehle, 2001) states "Highsmith transfers the CAS model to software development, viewing the development organization as the environment, its member as agents, and the product as the emergent result of competition and cooperation." ASD presents a state of vacuum in the beginning of its process. In other words, it doesn't set a specific tool to use. As it easily adopts, it drops. The following is an analogy from the Complex Adaptive Theory to ASD.

Table 2.7 ASD Adaptation of CAS Theory

Complex Adaptive Theory	Adaptive Software Development
Agents →	Developers
Environment →	Company
Emergence →	Product

ASD believes that a company is an environment and the developers are the agents. In the competing and dynamic environment, where the agents, or the developers, constantly interact and collaborate, a new product or the final project emerges. Just as

with other agile paradigms, ASD is largely based on RAD, but the iterative process is much faster and active than RAD (Highsmith, 2000).

Table 2.8 ASD Speed and Change Relevancy

	Low Speed	*High Speed*
Low Change	Waterfall	RAD
High Change	Evolutionary	Adaptive

The high change and speed of ASD "adopts" to any tools or technique that will bring the result. An example is ASD with XP's pair programming or ASD with SCRUM. The last stage, 'Learn,' is the time to look back and evaluate the experience. Besides being marketed for developing large systems, ASD is targeted to an environment that is under extreme pressure for meeting the delivery time, and that is facing high volatility and uncertainty. The result-driven rather than the process-driven is favored in the context of when the deciding factor is time. The high-value results are based on rapid adaptation to both external and internal events.

The key aspect in ASD is adaptation, and its nemesis is optimization. "Rather than the focusing on optimizing process improvement techniques, Adaptive Software Development emphasizes producing high-value results based on rapid adaptation to both external and internal events"(Highsmith, 2000). ASD does not believe that there is any value in optimization. The fundamentals of the traditional paradigms, techniques and framework are no longer able to do what they used to. Unlike the other agile paradigms, ASD is one agile paradigm that is suited for building large applications. Initially, ASD was born to service large applications. However, the ASD philosophy is certainly feasible in building mid to small size applications as well.

2.3.5 Scrum

Scrum is a hyper-productive development that achieves a very short product delivery time through the empowerment of the development team (Schwaber and Beedle, 2001; SCRUM website). Born out of a dissatisfaction of the waterfall model, the condensed four stages of requirements, design, prototype, and acceptance are executed continuously for each release, a sprint, in an average of thirty days or so. The term scrum is not an acronym. It's a rugby term that describes an act whereby everybody in the pack acts together with everyone else to move the ball down the field. The following is a Table that is used in describing scrum (Rising and Janoff, 2000; SCRUM website).

Table 2.9 SCRUM Terms Description

Terms	*Descriptions*
Sprint	One complete cycle that outputs a visible, usable, deliverable product
Backlog	An identification of all requirements that should be fulfilled in the completed product
Objects/Components	Self-contained reusable things
Packets	A group of objects within which a backlog item will be implemented.
Problems	What must be solved by a team member to implement a backlog item within an object(s)
Issues	Concerns that must be resolved prior to a backlog item being assigned to a packet or a problem being solved by a change to a packet
Solutions	The resolution of an issue or problem
Changes	The activities that are performed to resolve a problem
Risks	The risk associated with a problem, issue, or backlog item

A group of selected professionals are formed to deal with a project that is typically in chaos or out of control. Given the fixed completion time, the team works with a sense of urgency. Holding daily meetings and providing accountability for each member generates a number of sprints along the management of objects, packets, changes, and risks. At the end of each sprint there must be a visible and workable product. A series of these products would eventually lead to the completed project.

The key activity is the daily 15-minute meeting. In that meeting every member is responsible for delivering or reporting what he or she has promised the day before. In the case where a sprint would not produce a workable product, the team may drop a few functionalities instead of postponing the due date. Adhering to the due date is crucial. The two core ideas of scrum are team empowerment and adaptability. With team empowerment, the team realizes no obstacles in achieving its goal. If it sees any obstacles, then the management is called in for their removal. This is where the term "flip" comes in. It's the "flip" of the positions between the developers and management. Instead of the developers periodically reporting on the progress of a project, it's the management who comes to observe the progress and also supports the team unconditionally in resources.

Adaptability refers to the team decision-making for the next iterative direction at about every thirty days of the product release. This decision is based on what the team has accomplished and what the environment dictates. Another way of depicting scrum is that it's an exceedingly compressed waterfall model, utilizing incremental cycles and having at least twice the speed of the waterfall model. scrum believes that this is achievable when the empowered team, supported 100% by the management, is feeling confident in reaching the goal. It is a very humanistic method. It relies strongly on the

human factor; the management is obliged to eliminate all obstacles and listen to all of the team's requests.

2.3.6 Crystal

Crystal is a paradigm where one finds a project (from organized lists of projects sorted by project size and the number of people involved) that is similar to his/her situation and uses it by tweaking and modifying it. The idea originated from a software consultant who was, as a result of many years of consulting experience, able to create a large database of projects that is categorized per characteristics (Cockburn, 2001). Basically, the pool of projects is divided into two categories: the criticality of project and the number of people involved.

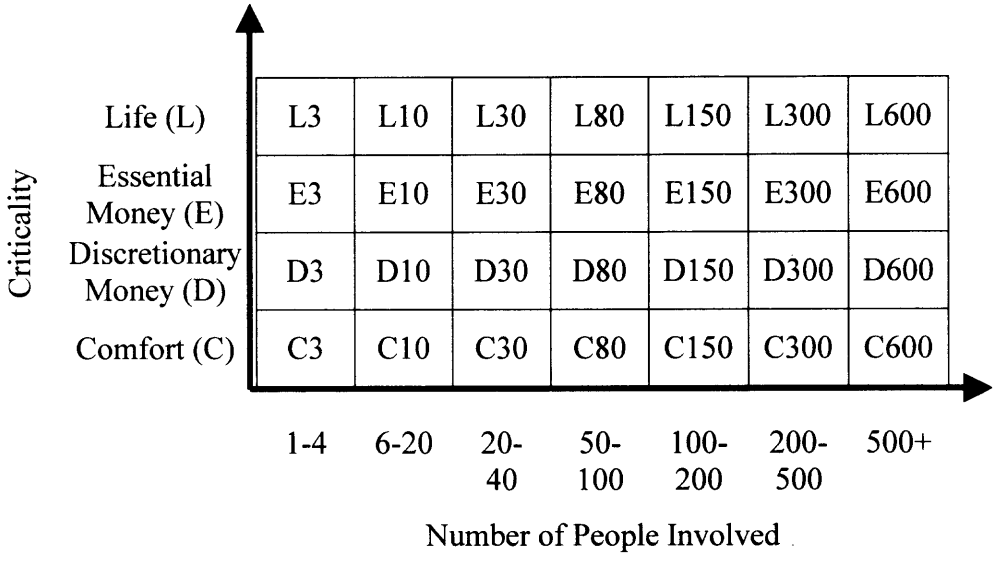


Figure 2.5 Crystal graph I.

The name, 'Crystal' is a metaphor. Crystal's clear to opaque colors and the mineral hardness scale metaphor help this paradigm to be easily remembered. The Y-axis, the criticality factor, is compared to the hardness of crystal, and the X-axis, the number of people involved factor, is compared to the color of crystal. Hence, the higher

the Y-axis value, the more critical the project is and the higher the X-axis, the more people that are involved in the project. Each Y-axis row indicates the level of project criticality by the project funding financial source. For example, 'discretionary money' is likely a project that would be used within a department whereby the project funding is managed within the department level. A cell reading "E80" refers to a significant and high-level project that carries an average of 80 professionals involved in the project. A friendlier graph is below.

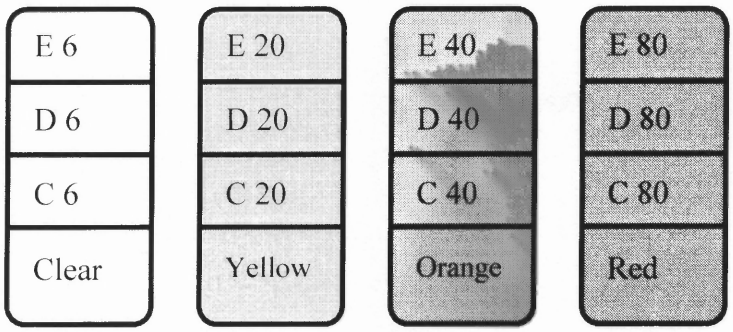


Figure 2.6 Crystal graph II.

In the business domain there is a case study. An organization's success or failure is carefully studied for the lessons to remember. Crystal offers a similar pedagogic approach. Each cell is a case study. By studying how that organization managed a problem, one can learn from it and apply the lessons to his or her project. Each cell or organization is displayed in the pool after a long series of in-depth interviews and observations. A typical flow of Crystal would be 1) decide and select a cell that the project may be a close fit for, 2) tailor and apply the techniques that were successful in the sample project, and finally 3) adjust and fine tune the techniques "on-the-fly." The key notion is "see what a successful project did and let's adapt it to our project".

Table 2.10 Core Crystal Elements

Two Values	People- and communication-centric	Highly Tolerant
Two Rules	Must use incremental dev.	Must hold pre- and post-increment reflection workshop
Two Base Techniques	The methodology-tuning technique	The technique used to hold the reflection workshop

Similar to other agile paradigms, the 'two values' indicate the flexibility, the 'two rules' indicate the iterative process, and the 'two base techniques' indicate the adaptation and the learning step. According to (Crystal website), Crystal gathers together a self-adapting family of "shrink-to-fit," human-powered software development methodologies based on these understandings:

1. Every project needs a slightly different set of policies and conventions, or methodology.
2. The workings of the project are sensitive to people issues, and improve as the people issues improve; individuals get better, and their teamwork gets better.
3. Better communications and frequent deliveries communication reduce the need for intermediate work products.

Living up to the agile paradigm principles, Crystal is a very flexible and carefree paradigm that allows the team to control, adapts, and chooses its path.

2.3.7 Dynamic Systems Development Method (DSDM)

Claimed as a framework rather than a paradigm, the origin of Dynamic Systems Development Method (DSDM) is traced back to Rapid Application Development (RAD) (DSDM website). It shares its birth with ASD, as both are the offspring of RAD. The explosive popularity of RAD led many organizations and consultants to jump on the RAD "bandwagon" and create their own proprietary RAD version. This caused many undesired problems and raised a number of concerns. After some assembly, DSDM, the "high-level" framework was born. DSDM has elevated and refined many basics of RAD.

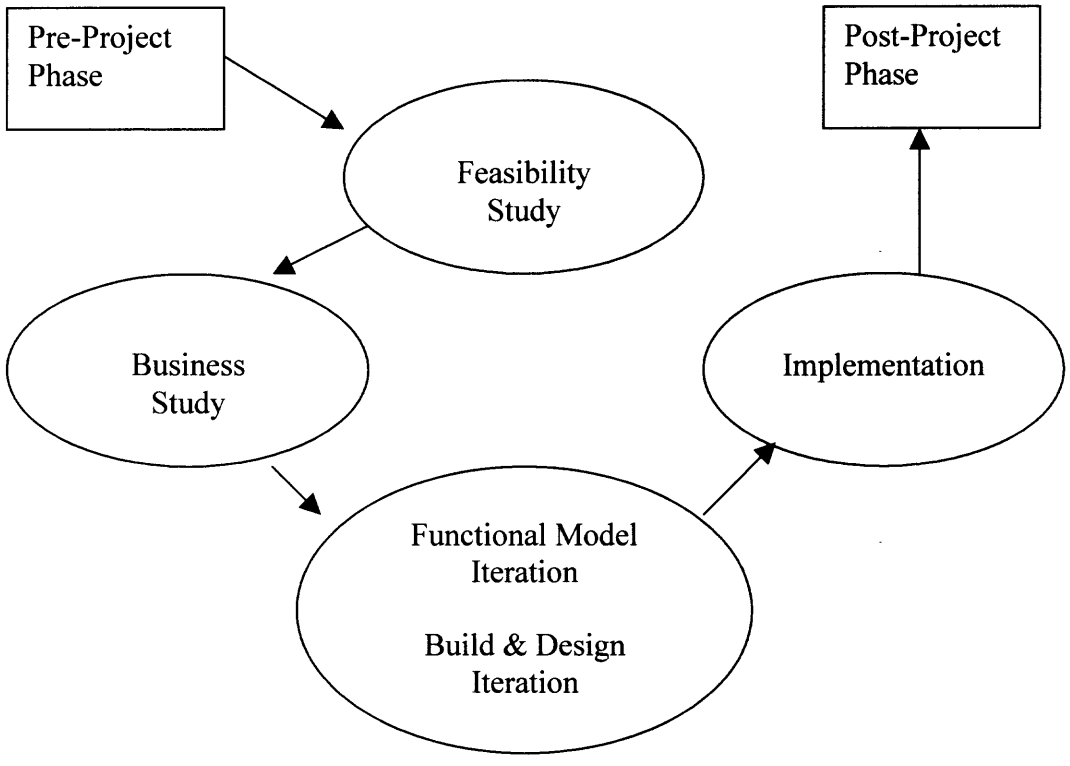


Figure 2.7 DSDM workflow.

Dynamic Systems Development Method consists of mainly four stages with each stage having a pre and post project phase, making it seven stages all together. The

engine of DSDM is in the Functional Model Iteration, Design, and Build Iteration stage. Like the RAD, this is where the continuous iteration happens. DSDM strategy is completely opposite to traditional approaches, such as linear sequential.

A typical project operates with a fixed requirements document, but DSDM operates with variable requirements documents, meaning the users may change or modify the requirements. This aligns it with all other agile software paradigms. In DSDM, the fixed items are the time and resources. Unlike the almost never-ending supply of resource and time elements in the traditional paradigms, they are fixed at the beginning of a project in DSDM. This is manifested in the following DSDM principles (DSDM website): 1) active user involvement, 2) team empowerment, 3) frequent delivery of products, 4) fitness for business purpose, 5) necessity of iterative and incremental development, 6) reversible changes, 7) requirements that are baseline at a high level, 8) integrated testing, and 9) collaboration and cooperation between all stakeholders. The overall theme is user-focused. Just like the other agile software paradigms, DSDM gives complete control to the users and accommodates unconditionally. As a RAD characteristic, DSDM is typically deployed in small to mid-size projects. A project is broken down for easy management and the iterative operation.

Like other agile software paradigms, DSDM may team up with other paradigms for a synergistic effect (i.e. DSDM with XP). The limitation of DSDM is inappropriate as it's still in its introductory state. However, based on the views of its principles, which are completely user-focused, it does draw a concern from the software developers. With more "ground" given to the users, there is less "ground" for the software developers to "stand on" and "work on."

2.3.8 Lean Programming

Lean programming or development is a concept that is entirely borrowed from "smart" manufacturing practices such as the Total Quality Management (TQM) and Just-In-Time (JIT) (Lean Prog. website 1). The philosophy was to bring the excellent quality and performance improvements that the manufacturing industry has enjoyed from TQM and JIT to the software process. This "superimposing" is certainly not a perfect fit, but both the manufacturing and software processes share common themes. To describe this, ten principles of lean programming are listed (Lean Prog. website 1):

- Eliminate Waste
- Minimize Inventory
- Maximize Flow
- Pull From Demand
- Empower Workers
- Meet Customer Requirements
- Do it Right the First Time
- Abolish Local Optimization
- Partner With Suppliers
- Create a Culture of Continuous Improvement

To "eliminate waste," the manufacturing organization carefully examines each step of its process for the elimination of non-value-added activity, which doesn't directly add anything to the final product. The stacks of documents that are generated during a software development may fit this bill. Similar to other agile software process paradigms, the bulky documents are considered "nice to have but not a must." The final verdict is that it simply doesn't have a direct line to the final product. In "minimizing inventory," the manufacturing organization's view is that the inventory consumes resources, slows

down response time, hides quality problems, gets lost, degrades, and becomes obsolete (Lean prog. website 1). In the software process, the software architecture or design is viewed as an inventory (Lean prog. website 2). For the complete analogy,

Table 2.11 Adaptation of The Manufacturing Principles

Manufacturing	*Lean Programming*
Eliminate Waste	Documents, diagrams, and models
Minimize Inventory	Excess documentation
Maximize Flow	Iterative development
Pull From Demand	Keep requirements flexible as close to system delivery as possible
Empower Workers	Empower Workers
Meet Customer Requirements	Iterative development
Do it Right the First Time	Write the tests first, and then write the code
Abolish Local Optimization	Maintain the high-level project objectives
Partner With Suppliers	Alliance with good suppliers
Create a Culture of Continuous Improvement	Lessons learned from earlier projects

For each principle of manufacturing, Lean programming has its corresponding principle. Again, similar to many other agile software paradigms such as ASD and DSDM, Lean programming is an iterative development, but it doesn't have a workflow as other paradigms do. There are no defined stages of Lean programming. Therefore, Lean programming resembles more of a framework than a paradigm.

2.4 Problems in Today's Software Development

Typically, the advent of a new software process paradigm is a solution to the problem that the previous paradigm is incapable of answering. Prototyping paradigm ease the sequential software process problem posed by Waterfall paradigm and the V-shaped software process paradigm alleviated the verification of the requirements problem. But, despite the uniqueness of many of today's software development paradigms and changing environment, time to time, a set of new problems surfaces along with the persisting issues. A true comprehensive list of all of today's software development problems and issues is definitely a challenge and treating it in here, just one section, would not be a just. Adhering to the purpose of this research, only a list of relevant problems and issues are addressed.

Requirements are a "moving target"

Maybe the most well-known and publicized problem in software development is that of constantly changing requirements (Borstler and Janning, 1992; Dorfman, 1997; Thayer, 1997). As one requirements engineering definition states "the area of knowledge concerned with communicating with organizational actors with respect to their visions, intentions, and activities regarding their need for computer support, and developing and

maintaining an adequate requirements specification of an information system" (Bubenko, 1995), this "many" actors, or the wide broadness of requirements scope complicate the problem. Ideally, when a final requirements document is signed off on, control of the development process is passed on to the developers. However, a situation characterized by continuous incoming requirements can severely hinder the progress of the developers and puts the process in a limbo. Also, in such a situation, the users may not initially know all of the features that they need and want in the new software (Gomaa, 1997). It may be well into the development process when the users discover the need for a new requirement.

Software Quality

Even with the implementation of software verification and validation (Wallace and Ippolito, 1997), many final software packages still exhibit a lack of quality (Basili and Boehm, 2001). One possible cause is the following. In many traditional software process paradigms, a working version of software only becomes available in the testing phase. Thus, in such software process paradigms, there is a greater probability that a major problem will go unnoticed and by the time such a problem is discovered, it is too late to recover in any economical sense (Sommerville, 2000; Von Mayrhauser, 1990; Gomaa, 1997).

Among the primary goals of testing is the evaluation of the software's overall performance and the verification of the software's stability and accuracy when integrated with other affiliated software or components. However, this single testing phase, as comprehensive as it is, tends to miss out on some unit level testing, and this can ultimately lead to an overall sub-optimal performance. Another possible cause for

poor software quality is that of poor documentation control (Han, 1994). In some cases, some prescribed requirements are not being implemented or are implemented incorrectly (Dorfman, 1997).

Difficulty understanding requirements document

Largely, this problem can result from either the document itself being poorly written or from the developer's interpretation of the document (Scharer, 1997; Frosberg and Mooz, 1996). In designing the requirements process, one relevant consideration is who prepares the document. In some cases, if a developer writes the document, he or she would observe and discuss with the user group prior to completing the document, but an inherent problem exists in that he or she is generating the requirements using a developer's viewpoint. Conversely, if an expert user from the user group is assigned this job, but he or she lacks the understanding of the developing tools' feasibilities and functionalities, then an impractical requirements document may be born.

'Runaway' projects/ Late delivery date and cost and resource overrun

A 'runaway' project may refer to a project, which is inundated with managerial problems and issues that there seems to be a vacuum in leadership and also, a low project completion percentage with the imminent delivery date. This phenomenon eventually leads to a non-linear cost reflecting the reworking of earlier phases of the development effort. Such a cost can be manifested in a variety of forms such as the exceeding of the project budget, the extension of the product delivery date, and reduced product functionality. Additionally, while it is widely accepted that change requests, whether trivial or mission critical, are an unavoidable component in the software

development process, the sheer number of change requests and the degree of their impact is excessive in a runaway project.

Lack of User Involvement/Participation

The need for strong user involvement in software development is well known and understood (Keen, 1981; Markus, 1983). However, the ability to practice it in every software development context can be a challenge because of a variety of factors. These include the characteristics of the user group, the organization's culture, the general attitude and perception of the users toward software development, and the users' past experience with regard to software development participation (Goguen, 1993). The two key hurdles are how to foster greater user assertiveness in their participation and how to structure the user participation in a user-friendlier manner so that the users can freely voice their opinions and participate (Bubenko, 1995). The responsibility clearly lies with both the users group and the developers group.

Social inertia/Resistance to change

In a software development context, social inertia refers to the difficulties of introducing a new system because of sociopolitical, managerial, and end-user impediments. In simpler terms, "no matter how hard you try, nothing seems to happen" (Keen, 1981) in the efforts of obtaining the acceptance from all of the stakeholders. The resistance to change may have several underlying motivations (Markus, 1983). Also, the resistance or managerial problem may lay in the closeness of the correlation of business objectives and software that supports the objectives (Bubenko and Wangler, 1993).

The steering committee's role

In almost any new software development, there is an assembly of people known as a steering committee who represent the interests of the groups that would be impacted by the new system. The role of the steering committee is critical to the software development process. The committee's responsibilities should include rendering decisions regarding all incoming change requests, handling emergencies, and coordinating all activities of the development (Kraut and Streeter, 1995). Problems may arise, however, when the committee plays a more passive role and functions only as a messenger whose sole responsibility is disseminating information among the affected parties. This scenario may happen since the role of the steering committee is variable and is influenced by the intricate links and relationships between the developers group, the committee, and all of the involved parties. It is these links and relationships that eventually mold what the role of the steering committee would be.

Standardization of tools

Software development operations continue to become more global but also must not lose sight of local customer demand (Stalk & Hout, 1990; Taylor, 1992; Battin, et al., 2001). This forces organizations to adopt systems that can accommodate both central and local operations (Carmel, 1997). However, this multi-site development can give rise to several problems (Carmel and Agarwal, 2001). Problems involving managerial issues can occur as well as problems with standardization of tools (Ebert and De Neve, 2001). The preference of tools may vary according to different parts of the world. Moreover, the availability of tools and resistance to certain tools by local developers may compound the problem.

2.5 Implications of Contemporary Paradigms

All of agile software process paradigms exhibit the similar attributes that when they are compared to the problems above, they all positively influence the problems. This is because the agile paradigms were born from the same spirit and philosophy (Agile software manifesto website; Cockburn, 2001; McCauley, 2001). They all have the same common goals, which are no overhead, no baggage, flexibility, and the fastest possible product delivery (Abrahamsson, et al., 2003).

The differences among the paradigms, which mostly center on the formation differences in the cycle, do not alter the paradigms' goals. Only the document control problem is perceived to be negatively influenced by all of the agile software process paradigms. In contrast to the traditional paradigms, the agile paradigms view software documents as being of low value. Also, the agile paradigms theme is "no baggage."

Conspicuously, the agile paradigms and the traditional paradigms sit at the opposite ends of the software development paradigm spectrum. However, it is important to also note that many agile software process paradigms are incapable in projects that most of the traditional software process paradigms can (Abrahamsson, et al., 2003), i.e. mission critical system or large management information system. Below, a more detailed analysis of the agile software process paradigms is discussed.

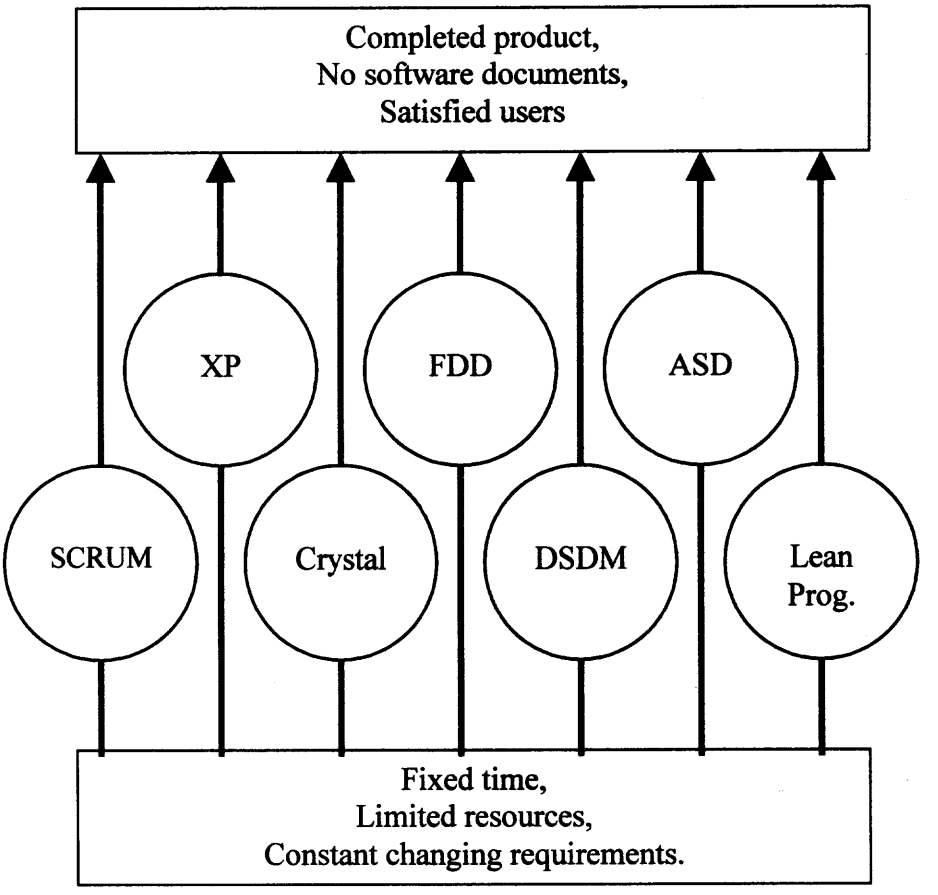


Figure 2.8 The shared path of agile paradigms.

Table 2.12 Agile Software Process Paradigms Characteristics

Constraints	XP	FDD	ASD	SCRUM	DSDM	Lean Programming
Be able to go back and fix?	√	√	√	√	√	√
Need a set of completed requirements to begin?						
Accommodate "live" requirements?	√	√	√	√	√	√
A set of documentation generated?						
Flexible to change requests?	√	√	√	√	√	√
Produce a working product in a short time?	√	√	√	√	√	√
Used on large scale projects?			√			
Used on mission critical projects?						
Self-contained developing teams?	√	√	√	√	√	√
Active User Involvement?	√	√	√	√	√	√
Easy to up-scale the project after its release?	√	√	√	√	√	√
Good for legacy systems?			√			
Has a road map for project?		√	√	√	√	
Possible to integrate with other paradigms?	√	√	√	√	√	√
High quality product?	√	√	√	√	√	√

All agile paradigms possess very similar themes. The most outstanding ones are the iterative process, constantly changing requirements, and high tolerance level. They are very forgiving and willing but with an extreme sense of urgency. A typical agile paradigm project typically has a fixed time, limited resources, and constantly changing requirements. At the end, the agile paradigms deliver the final product in time to the very satisfied users. The software professionals are equally satisfied working with such tolerable paradigms and with no documentation. Here is a summary of the characteristics:

- Human factor is the key
- Collaboration, communication, and cooperation
- Adaptability is the life
- Fast and short iterative cycles
- No "division of labor," everyone does everything TOGETHER
- Direct Face-to-face contact (continuous feedback)

Following different dimensions of agile paradigms are discussed:

1) Newness of technology
2) Risk
3) Implementation
4) Impact on customers

• Newness of Technology

The newly discovered agile models are still being established. As manifested by the term itself, the ideology of agile software development is being discussed and is still unsettled formally (agile software manifesto website). While the theorists are busy defining and figuring out the new wave of software development, the practitioners are already using them (Beck and Fowler, 2001; Beck, 2000; Highsmith, 2000; Cockburn, 2001). As the newness has it, each agile model is unique in its own way and different. They are different in "how to," but the objectives are same. The shared objectives are: 1) focus on the customers and interactions, 2) response to constant changes, 3) non-stop iterative coding and testing, 4) and collaboration with customers in the whole process (Abrahamsson, et al., 2003).

The agile models are in a par with the business mission of today's time, focusing on the customers. "Customer-friendly" has become the mission statement for many organizations regardless of the industry type. Led by e-commerce, the software development process is obliged to be ever more customer-friendly in its product and the process. The traditional software development models embraced customers only during the feasibility study and the requirements solicitation. Newness is indicated through the volume of new publications of agile models (Beck, 2000; Cockburn, 2001; Beck and Fowler, 2001; Williams, et al., 2000; Riehle, 2001; Highsmith, 2000), which tells that there is a significant market demand and desire to adopt agile models.

• Risk

Among the protruding characteristics of agile models, following is a list of the general risk areas: 1) negotiated quality, 2) early coding, 3) frequent releases, 4) parallel

development, and 5) out of control. With the quality concern (Baker, 2001), the "negotiated quality" (Baskerville, et al., 2001) is a win-win situation for both the customer and the developers.

Extreme programming exemplifies this "negotiated quality" with the on-site customer. The customer's demand of quick product release matches the developer's goal of customer satisfaction. But maybe the actual product end-users are lost in this negotiation. Driven by the "time-to-market" phenomenon and the blazing speed of e-commerce, this risky practice is likely to continue. However, there is the consequence of low quality (Cosgrove, 2001). The dilemma is finding the optimal and maximal quality level.

Early coding is only a problem if people's perception of the requirements gathering process is unchanged. Frequent releases are a way to meet the fast growing customer demands and responding to the shortcomings or the quality issues of the end products. This may downgrade the significance of each release. Demanding customers can be somewhat harnessed and "controlled" and the developers' productivity is justified, thus providing the ground for the developers' escape route such as "well, we'll put it in the next release" or "we'll just fix it in the next release." Parallel development is an ideal concept, but "how" is the question.

In Extreme programming, it features pair programming, but this may threaten the very factors that make the programming such a rewarding and enjoyable task (Glass, 2001). Another risk would be the "no documentation" or minimal documentation. This refers to the elaborate user requirements, software and system requirements, test protocol, and other relevant documentation. The development cycle isn't completed when the final product is delivered into the hands of the users. Maintenance and care

taking certainly take equal, if not more, effort. With no development document such as software architecture, design and detailed user requirements, it is a challenge to complete any maintenance or modifications. Being able to read the "agile-version" document (XP's documentation is the code itself) and complete an assignment is still a question to be answered.

• Implementation

The implementation is probably the most challenging part in this new wave of paradigm. Everyone wants to find out how to implement XP, ASD, SCRUM, or Crystal, because they are so new. Organizational structure is one thing, but the complete new schooling of this new approach in the minds of traditional developers is not an easy task. It's almost as drastic as teaching a left-hander how to do it as a right-hander. No pre-defined requirements, continuous testing, and early coding are hard practices to absorb for the developers.

For the implementation, a whole new change in working environment is inevitable. Extreme programming asks for an open working space. The rigid, structured, and pre-defined boundaries no longer exist, and there is more freedom and flexibility to experience the spirit of agile models. Usually, this translates into a small size team, a team that is composed of creative minds, and well-trained and able personnel. The key concept of the agile model is customer satisfaction. But it's also the most difficult thing. The agile model attitude on customer satisfaction is to let the customer "build everything" and to let the developers just "give a hand." The customer must be on site with the developers, as all works must be done according to the customer's view as exampled in extreme programming (Beck, 2000). Manifested by the attributes, the

products' most concerned factor is the customer satisfaction. ASD (Highsmith, 2000), which deals with developing such product, adopts the philosophy of "speculate, collaborate, and learn."

• Impact on customers

Agile models probably deliver the most customer interaction. Driven by the customer, the agile model's spirit of "quick-to-market" pervades all agile model features. Extreme programming (Beck and Fowler, 2001; Beck, 2000) has a customer on-site; listening to the customer is one of XP's "four basic activities." It would not be an overstatement if the development is led by the customer. ASD (Highsmith, 2000), mostly based on RAD and evolutionary prototyping, also embodies heavy customer interaction through iterative cycles. The final software product that comes out of an agile model differs from the software products from the traditional software development channel in terms of the development experience. The attitude is different, the process is different, and the perception is different. The customer-focused agile model tries to incorporate the customer's requirements into the product as much as possible.

Each paradigm is reviewed for its uniqueness and dramatic differences from the traditional paradigms. Propelled by their agility, the benefits and results are refreshingly striking. Initially, the obstacle is to gain the confidence of users, especially the senior level professionals who have spent many years with the traditional paradigms. Because they are new, the models' long-term values can be reassessed. For example, Extreme programming said documents do not carry any value, at least in the short-term. What can happen if XP developed software requires re-work or troubleshooting and there are no documents? To make the situation worse, what if the original developers are no longer

with the company? How can XP explain this? From the management viewpoint, what can be observed to be different in using an agile paradigm? Will the difference in the product delivery cause a change in the environmental factors? How about the developers' perception? Where is the new bottleneck? The ripple effect of agile models will certainly make the surrounding environment be noticed.

The emergence of agile models undoubtedly made the traditional models less visible. Adding to that phenomenon is the market. Led by e-commerce, the software market demands quick, easy, and short-life software. What does this tells about the traditional models? What is the future with them? How will they evolve or diminish? Unquestionably, many traditional models are still robust in places like the military and other government-regulated industries. Yet, the software community is embracing agile software development as one of its main staples. As new as the models are, the inner world of agile models presents unlimited research opportunities. Each model is "in-progress" in terms of reaching its full potential. Each fine inner point of an agile model carries its implications to the surrounding underlying factors.

CHAPTER 3

PAIR PROGRAMMING HYPOTHESES DEVELOPMENT

3.1 Pair Programming and Extreme Programming Defined

Pair-Programming (PP) can be defined as a programming paradigm where two individuals program sit next to each other sharing one PC, one monitor, one keyboard and one mouse. This is different than 'mutual programming.' Mutual programming can be described as a programming practice where a programming task is divided among the programmers and each programmer programs his or her share on a separate computer.

It would be a common scene in mutual programming for two programmers looking over a monitor and discuss and collaborate but not during the entire task. PP truly asks two programmers to work as one; PP can be pictured as "two-headed one body" programming.

Conveniently, the programmer who is behind the keyboard is label the "driver" and the other programmer the "navigator." However, other terms or labels can also be used to describe. Intentionally and conspicuously, the roles of both programmers are not clearly defined nor separated, but typically the "driver" drives or types the keyboard for coding and the "navigator" communicates and collaborates with the "driver" in designing, coding, debugging and testing.

Defining the exact role definitions would traverse the XP and PP spirit (Beck, 2000). The list of PP benefits are quality, morale, trust and teamwork, knowledge of transfer, enhanced learning and more (Williams and Kessler, 2002), but probably the biggest immediate benefit would be another set of eyes looking over acting as the double "net" catching any flaws and debugging. The role switches and the switch frequency are to the two programmers discretion.

Pair programming is mainly used for a small or maybe up to a mid-size application development that are risky projects with dynamic requirements. Used in extreme programming (XP), PP stretches the coding task to incorporate code designing, debugging and testing. PP is not a software development process like a linear sequential paradigm, rather it's considered to be a stage. It's an extensive stage where many activities are spontaneously performed as mentioned above. All these anecdotal and unconventional characteristics make what PP is and why it is so effective.

User participation is a significant issue in software and system development, however one must clarify that the programmers of PP are not the users nor end-users. In this work, the PP programmers are the software development professionals or engineers who are devoted solely to developing and delivering software or systems to the targeted users. The programmers are simply paired for the purpose of practicing PP and attaining its benefits.

The area of programming support tools is another significant area that impacts and also complements the programming activity (DeFranco-Tommarello and Deek, 2003). The presence of these programming support tools is illuminated in many collaboration projects. In the PP domain, the programming supporting tools allow two individuals of geographically different locations to perform PP. This virtual PP mode benefits organizations with global operations of software and system development.

In the light of recent streams of agile software development approaches (Cockburn, 2001), PP of extreme programming (Beck, 2000) brings surprising and promising result (Nosek, 1998; Williams, et al., 2000; Williams & Upchurch, 2001; Williams, 2000; Williams & Kessler, 2002). A study shows a very high level of practitioners' satisfaction (Williams, 2000).

Given these favorable results, an inevitable question comes to one's mind; "Can we make it better?" In order to continue this push of PP, a further study is needed. Given the fact that two heads programming with just one keyboard and one monitor, which is still the fact that is difficult to accept by some people, the psychosocial probing deserve some consideration for meeting the objective. Besides a general work of psychology of programming (Weinberg, 1998), the specific 'pair' phenomenon of PP is being focused. Therefore, the objective is an effort to gain an deeper understanding of the facts that underlie the 'pair' phenomenon and influence PP productivity.

In this section, a careful review is scheduled on what prior PP research has been done; description of the details of PP preliminary field survey that was conducted in collecting substantial evidence; and lastly personality and communication concepts that are consequence of the survey result. Most importantly, a clear understanding of the prior PP researches would lay a good solid ground to keep this experiment in track.

The field survey is a complement and supplement to the prior PP works. The fact that the survey is from a group of very experienced practitioners, adds more value to the cause. A review of personality and communication concepts is a direct result of the survey. As the details would be covered in its appropriate section, personality and communication among many other factors appear to have significant effect to PP productivity.

3.2 Pair Programming Literature Review

Currently, known as one of twelve core practices of XP, PP has been around (Constantine, 1995) and not a XP invention. However, PP rose to the spotlight in the wave of XP and other promising agile software process paradigms (Cockburn, 2001).

There had not been a whole lot of studies in PP general until recently. Even now, the effort is just beginning (Williams and Kessler, 2002). The first successful large-scale empirical work of PP, and also the showcase of XP, is "C3" team of Chrysler project (Anderson, et al., 1998). In a real situation of chaos, by deploying PP, C3 team has managed to put a runaway project on the right track within the expected completing time. This case study is significant as it's a very large subject pool and a real industrial situation. Another experiment with professionals, a 45-minutes long lab-controlled experiment (Nosek, 1998) was conducted where the fifteen programming professionals were divided into either PP groups or one-man programming groups. In the experiment, PP groups have generated 1) a longer completion time, 2) a higher quality work, 3) a higher enjoyment and confidence in the work, and 4) experienced programmers performed better.

Using university students as subjects, a few XP controlled-experiments (Muller and Tichy, 2001; Kivi, et al., 2000) in university environment were performed. A small scale (only one team, 4 subjects in a team) a year long lasting student XP controlled-experiment reported a negative PP experience (Kivi, et al., 2000); four months into the experiment, PP was dropped by the subjects because "team members felt that paired programming was a waste of time". The authors contributed the reason to subjects' no prior software development experience and the difficulty of finding common hours to pair-up. The subjects were also given a freedom to pair-up at their discretion.

A more controlled PP experiment consisted of 41 student subjects (Williams, 2000; Williams, 2001) was conducted. Besides the main objectives, the experiment also looked at possible impacts of grade point average, gender, and personality types to the outcome. The experiment had seven male-male combination teams, four male-female combination teams, and one female-female combination team. Contrary to the common expectation of "get the job done in half of the time," most empirical works showed slightly longer completion time than one-man programming mode. Nonetheless, PP did exhibit a high level of quality (much lower error rate), programmers' satisfaction and confidence, which are significant factors in sustaining and completing long successful projects. A discussion of pairs with different levels of programming skill and pairs of different combinations of extroverts and introverts were observed, but with no conclusive evidence of empirical data (Williams and Kessler, 2002) are available.

Table 3.1 Summary of Pair Programming Studies

Studies	Authors
Case Study	Anderson, et al., 1998
Lab-controlled experiment	Nosek, 1998
Lab-controlled experiment (XP)	Kivi, et al., 2000
Lab-controlled experiment	Williams, 2000
Lab-controlled experiment (XP)	Muller and Tichy, 2001

In an attempt to classify PP, the team-programming concept is briefly visited. There are many different team concepts, just to name a few: 1) decentralized (egoless) team, 2) centralized (chief programmer or surgical) team, and 3) controlled decentralized.

The decentralized team (Weinberg, 1998), characterized by generally having ten or fewer programmers in a team, shares coding activity, decision-making, leadership, problem solving, communication, or any other group activities. It is reported that the decentralized team is an ideal programming team for long-term projects with minimal time constraints, but not for urgent projects (Mantei, 1981; Shneiderman, 1980). It also enjoys a high level of satisfaction (Shaw, 1971) with shared decision-making and open communication, but exhibits riskier behavior because of the dispersion of failure (Mantei, 1981).

The centralized team, or popularly referred to as the 'surgical team,' transports all team activities by the linear up-and-down channel (Jurison, 1999; Rettig and Simons, 1993; Mantei, 1981; Unger and Walker, 1977) Typically three members, the chief programmer, programmer, and program librarian, form the centralized team. However, additional members can be included per specific project needs. The centralized team is more suited for projects with time constraints and is more capable than decentralized team of meeting the final product delivery date. Some of its limitations are low satisfaction, morale, and group cohesiveness (Shneiderman, 1980).

A concept that falls between decentralized and centralized teams is that of the controlled-decentralized team (Mantei, 1981). It consists of a project leader, senior programmers, and groups of programmers. A senior programmer manages a group of programmers, and the senior programmer reports to the project leader. There are also many senior programmers. The team is structured like a centralized team, but functions

and behaves like a decentralized team. The controlled-decentralized team is best suited for short term, large-scale projects.

3.3 Pair-Programming Field Survey

Along with a literature review, information obtained from a field study brings substantial value to a study. This preliminary survey addresses the discovery of influencing factors of PP productivity from a group of very experienced PP professionals. It's their personal accounts, experiences within their capacities and perspectives that makeup the survey result.

Searching for the widest breadth of the various careers, positions, experiences and industries of PP professionals, Internet was selected and used. Most online groups were under XP topic category and were named either by their geographic locations or XP applications. For example, typical group names were "chicago-agile-dev" for Agile Development group in Chicago, "xptoronto" for an XP interest group in Toronto, Canada, and "xp-at-school for XP in classroom. Message thread asking survey participation was posted. In creating the survey questions, besides being a brief one, the expectation was to solicit the professionals' opinions. A short list of possible PP productivity maximization success factors (or impact variables) and provisions for any new additional variables that the participants may want to enter were given and provided. Beginning with a set of career profile questions, the main section of the survey is the variable list. In making up the list, a set of common variables is put together. They are as follows;

- Gender
- Programming skill
- Cognitive/Programming style
- Personality
- Familiarity
- Fluency in English (Communication)
- Pair Protocol

Following is the explanation of the variables. 'Gender' - Some earlier works speak of "cross-cultural communication" between men and women (Stewart, et al., 1996; Tannen, 1991). The belief is that there is an intrinsic difference in communication between men and women, largely because "from early childhood, girls play with a best friend or in a small group and use language to seek confirmation and reinforce intimacy, whereas boys use language to protect their independence and negotiate status in large-group activities" (Tannen, 1991). This variable choice is to see how a professional perceives the gender variable as a difference in PP productivity based on his or her own PP experience.

'Programming skill' - An obvious choice, this is to observe what degree of impact does this variable bring in PP productivity. 'Cognitive/Programming style' - Programming is perceived to be an individual's creation that is very reflective of his or her problem solving approach (Wiedenbeck and Ramalingam, 1999; Corritore and Wiedenbeck, 1999; Brooks, 1999; Engebretson and Wiedenbeck, 2002; Fix, et al., 1993; Mosemann and Wiedenbeck, 2001; Deek and McHugh, 2000; Deek, et al., 2000). With

its subjectivity, the collaboration of two very different or very similar styles of individuals in PP context is an interesting research question.

'Personality' – Pair programming is not only a programming activity, but a social activity as well (Yourdon, 1997; DeMarco and Lister, 1999). The general consensus is that a typical programmer is 'detached' personality type (Weinberg, 1998). In PP where collaboration and "pulling for each other" are essences, it would be interesting how this 'detached' personality interact or be 'attached.' It just might be that psychosocial aspect is a greater influence in PP. In this survey definition, personality not only includes the cognitive aspect, but it also includes other dimensions such as temperament, demeanor, personal values, and attitudes.

'Familiarity - This addresses the comfort level that is generated by two programmers. It can be defined broad as being "buddies," worked together many years, and have similar background. 'Fluency in English' or Communication - The face-to-face communication between a pair is perceived to be, maybe the single most important factor in creating collaborated good. 'Fluency in English' variable is chosen because the questionnaire is done in English and mostly English speaking professionals in English-speaking countries would participate the survey. However, the true underlying meaning is communication. The question is how much of and significantly does communication factor impacts PP productivity.

'Pair Protocol' - This refers to a set of structured or semi-structured guideline for pair programmers to conduct during PP besides the continuous designing, coding, and reviewing. Executed in an unsubtle manner, it would be a list of action items that directs the pair to adhere besides the continuous designing, coding and reviewing. One example is how long one should "drive" the terminal and mouse and how often the pair should

switch being the "driver." In previous few empirical XP studies (Kivi, et al., 2000; Muller and Tichy, 2001) a desire to have a set of guideline of "how to" in pair conduct was hinted. Appending the variable list, a provision was made for the participants to enter any variable(s) that they believe it makes an impact.

3.3.1 The Survey Participants Profile

All together, 44 responses were received either through email or as posted messages in the online message board. With a confidentiality concern, most of responses came in via email and only two responses were posted as message thread. Reading by the names of the groups and email addresses, vaguely but a good estimation is that the responses came in from many different parts of the world, from nearby Canada to far from Europe. The diversity of participant pool certainly has meet one of the survey's objectives. Following summary charts of participants' profile are provided.

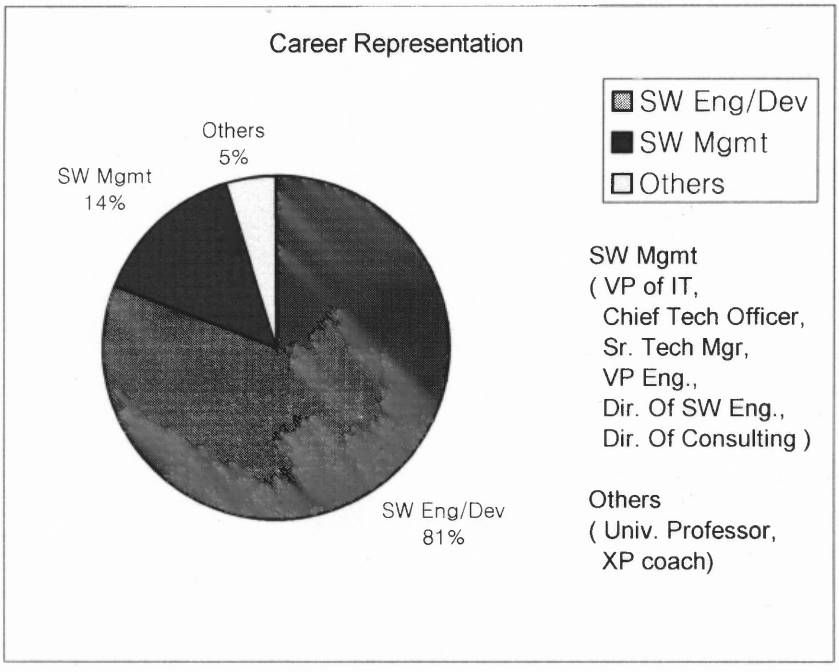


Figure 3.1 Career representation.

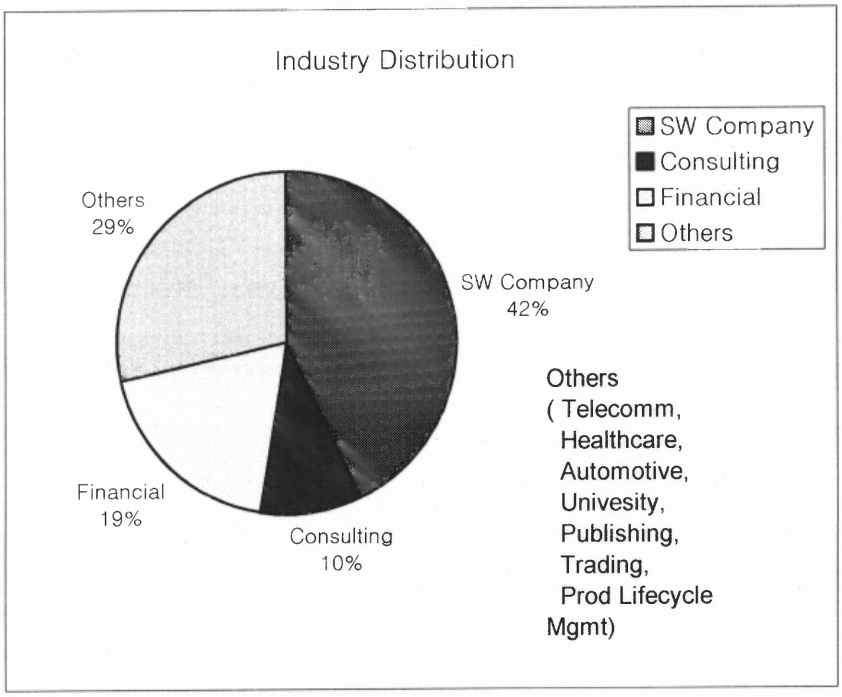


Figure 3.2 Industry distribution.

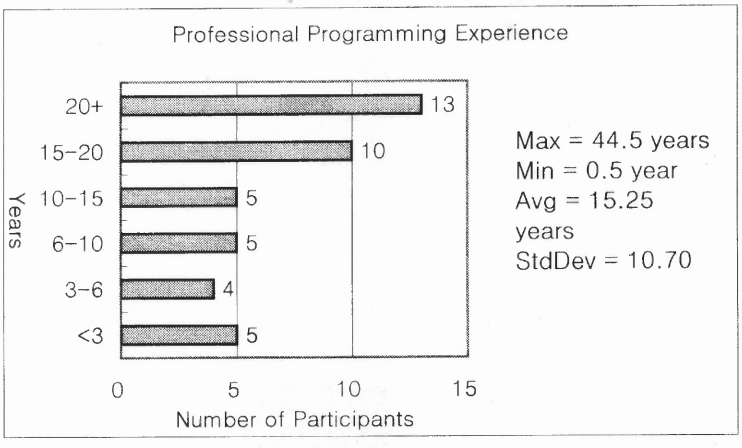


Figure 3.3 Professional programming experience.

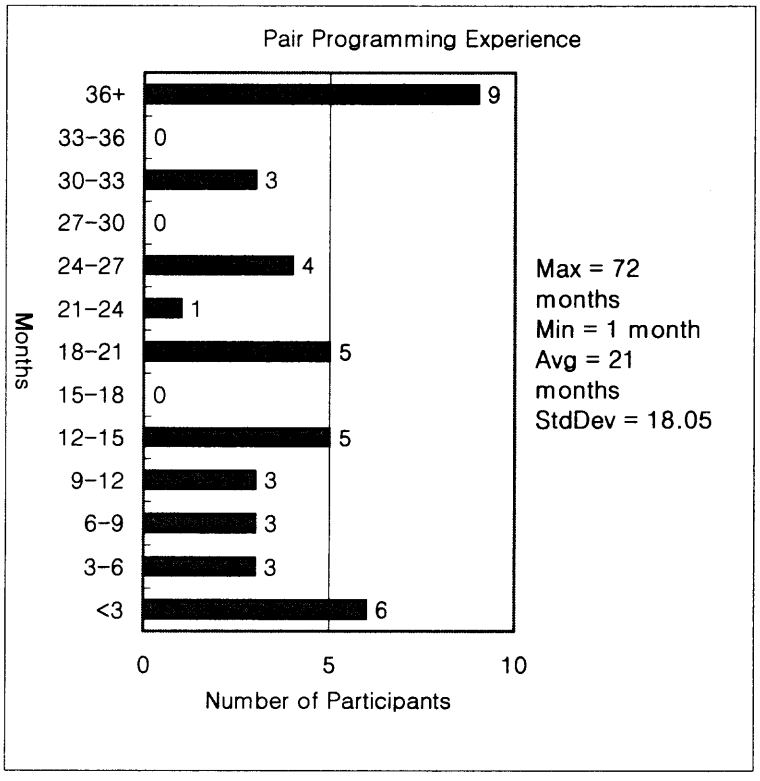


Figure 3.4 Pair programming experience.

On average, over 15 years of programming experience and 21 months of PP gives a very experienced group. In the career representation, software developers or engineers and a share of management are mostly represented. Probably the most diverse area of the profile is the industry distribution. Lead by the software development and manufacturing industry, a considerable financial, mainly banking, industry and a long list of various industries round up the distribution. Cumulatively, a typical participant is depicted as a software developer working in a software development organization possessing a near two years of PP and 15 years of professional programming experience. The gender distribution is predominantly male as only five females are represented out of total 44 participants. Out of all responses, two responses only completed the comment section.

Many have indicated an interest of this survey's outcome, which shows the survey's appropriateness.

3.3.2 The Survey Result Analysis

In decoding the submitted data, there were some difficulties. One example is a case where the participant deployed his own ranking system by assigning "High Impact," "Medium Impact," and "Low Impact" instead of assigning algebraic number one through seven or higher. A couple of responses ranked 'gender' variable with a number 99 and 1000 each, as an exhibition of a strong disagreement with the 'gender' variable. Assigning same ranking number to two or more variables was observed. In many cases, some variables were not ranked at all. Along with the ranking, a high interest was new variables from the participants. In the questionnaire, a blank provision is provided where the participants would fill them out with new variables. Following is the new added variable list;

- Technical knowledge
- Business domain knowledge
- "Keyboard switch"
- Ability to work with others
- Willingness to communicate
- Personal hygiene
- Working environment
- Coding conventions/Platform
- Time constraint
- Availability of snacks

- Open to new ideas
- Experience
- Common expectation
- Tiredness
- Desire to learn

Every survey participant did not review these new variables which have been added by some of the survey participants. In other words, not everyone had an opportunity to review these new variables and rank them. However, the importance of the survey is reflected more in the collection of a list of influencing variables, and not in the variable ranking order.

Interestingly, many psychosocial variables such as "open to new ideas," "common expectation," "desire to learn," "ability to work with others," and "willingness to communication" appeared and they outnumbered the explicit variables such as "coding convention," "same operating system platform," and "working environment." The following chart is the summary ranking result.

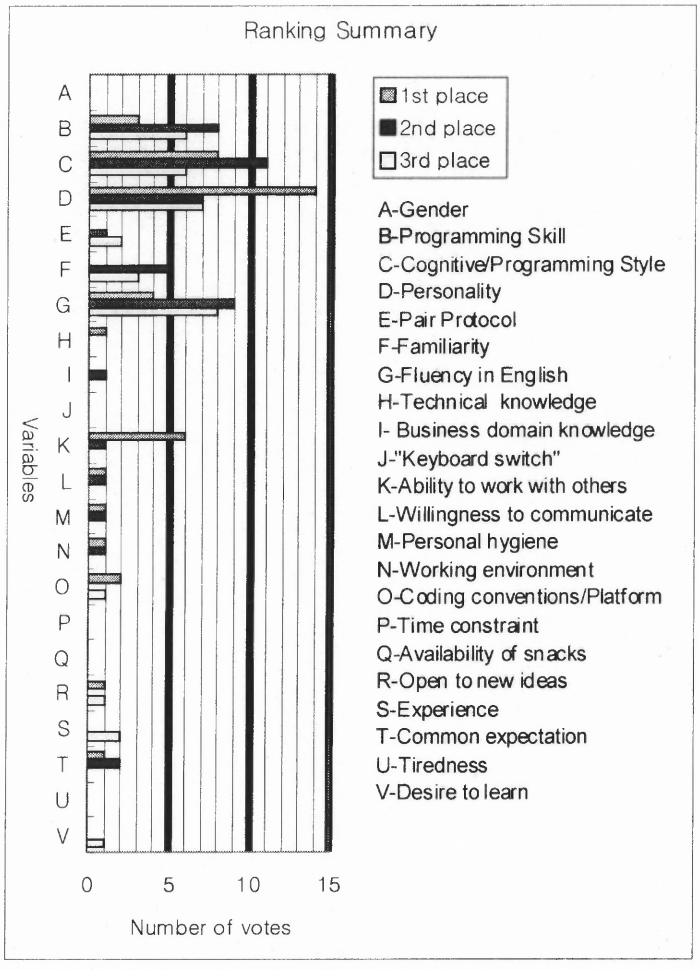


Figure 3.5 Ranking summary.

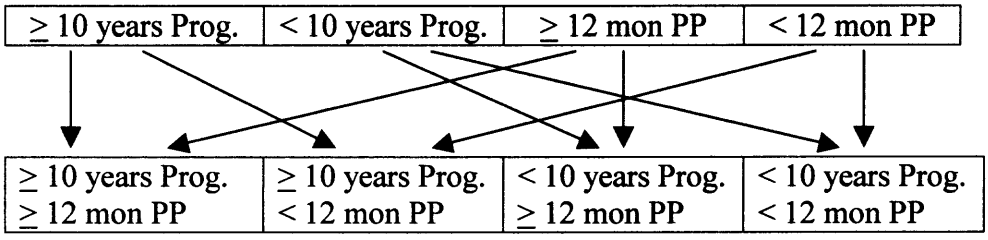
The legend of 1st place, 2nd place and 3rd place means how many 1st, 2nd, and 3rd place votes for that variable. Some variables that have no bar graph would mean that it received 4th place vote or beyond. Again, A to G variables are the original variables and H to V variables are the added variables by the participants. From the result, D, 'personality' variable is the most influencing variable which received 14 1st place votes, seven 2nd place votes and seven 3rd place votes. This is followed by C, 'cognitive/programming style,' G, 'fluency in English, and B, 'programming skill.' K, 'ability to work with others' is an interesting appended variable which received six 1st place votes and one 2nd place vote. In a larger picture, a close relationship between

'personality' and 'ability to work with others' can be assumed. There are total 12 variables that received at least one 1st place vote, 12 variables that received at least one 2nd place vote, and 10 variables that received at least 1 third place vote. There were five variables that did not receive any top place vote (1st, 2nd, and 3rd place). The overall picture of the ranking summary suggests that there is rather a large issue of teamwork (trust, willingness, and open-minded). The popular variables that most professionals chose as the PP productivity influencing variables are mostly psychosocial variables. It wasn't any controllable physical factor, rather it was "how much WE can put our efforts together" to reach a goal. That "effort" is manifested through variables such as 'open-minded,' 'willingness to communication,' 'ability to work with others,' 'common expectation,' 'desire to learn,' and 'personality.'

The number one choice of 'personality' reinforces the view of "you have to be nice to your playmate" (Williams & Kessler, 2000). Every programmer has his or her own programming style and habit, hence compromising and reconciling this personal inclination within PP environment is a constant challenge. The 'gender' variable consistently has been voted to be the last or be the least influencing variable. Surprisingly, 'personal hygiene' received one 2nd place vote and one 3rd place vote. Being almost lap-to-lap for the whole day, a certain courtesy level of personal hygiene deemed to be a requirement in PP.

A very diverse participant pool as this is, a further data breakdown and analysis may uncover more hidden facts. Where experience influences judgment, an observation is made on what kind of pictures is drawn if the data is further broken down under two sectors of "more experienced" and "less experienced." The difficult part is that what constitutes "more" and "less" as these terms are very subjective and a context specific.

But for this research's purpose and based on the profile, a group of ten or more years of programming experience as "more experienced" and less than ten years for "less experienced" are chosen. Similarly, twelve or more months in PP is labeled as "more experienced" and less than twelve months in PP as "less experienced." As a step further, four additional ones are selectively grouped as the following figure is depicted. These four additional groups are to observe any saliencies from the association of PP and general programming experience in a relation to the PP productivity maximization.



The following figures (eight of them altogether) are the variable ranking display more than eight groups.

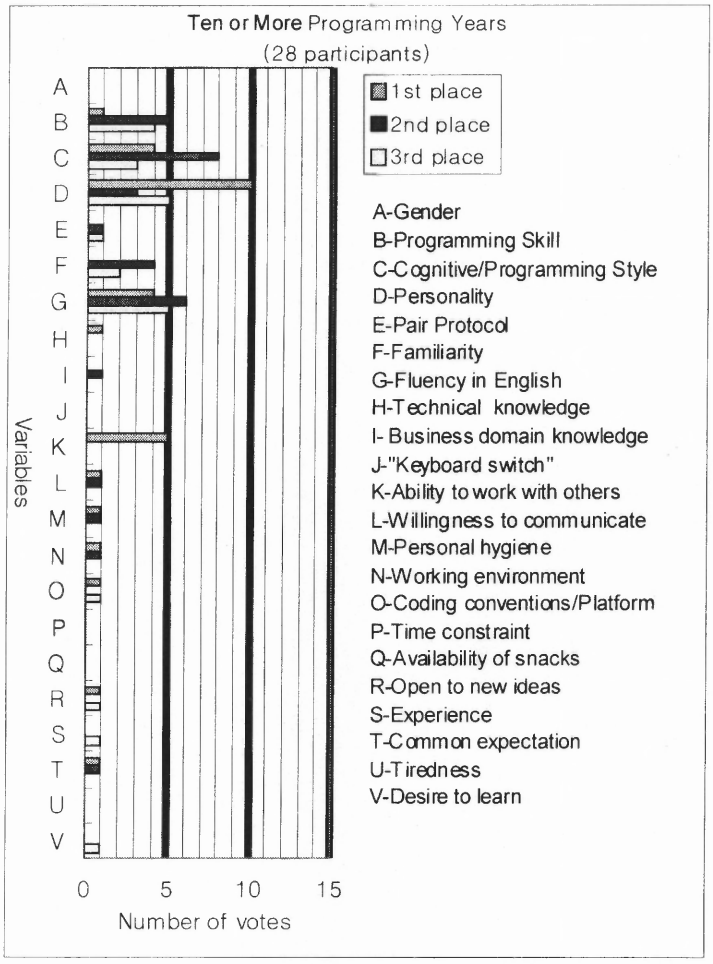


Figure 3.6 Ranking result from the subjects with ten or more programming years

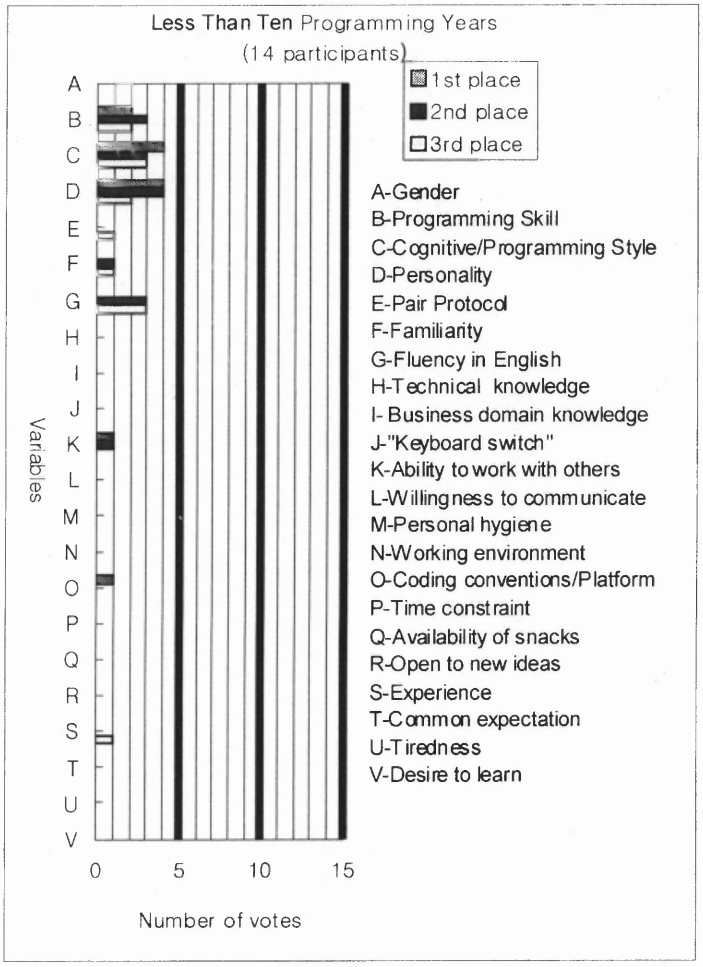


Figure 3.7 Ranking result from the subjects who possess less than ten programming years.

In both groups, the variables 'programming skill,' 'cognitive/programming style,' 'personality,' 'fluency in English' are popular choices with 'personality' receiving the most votes for first place. Contrast to "less experienced" group, the "more experienced" group shows the 'personality' variable dominating the 1st place votes along with the 'ability to work with others' variable. The "more experienced" group also had more added variables and many of them received 1st, 2nd or 3rd place votes. 'Ability to work with others' variable has five 1st place votes. A possible explanation of this observation is that the "more experienced" professionals have a deeper root

understanding and insights of PP and also their maturity level allows them to comprehend the surrounding environment of PP better.

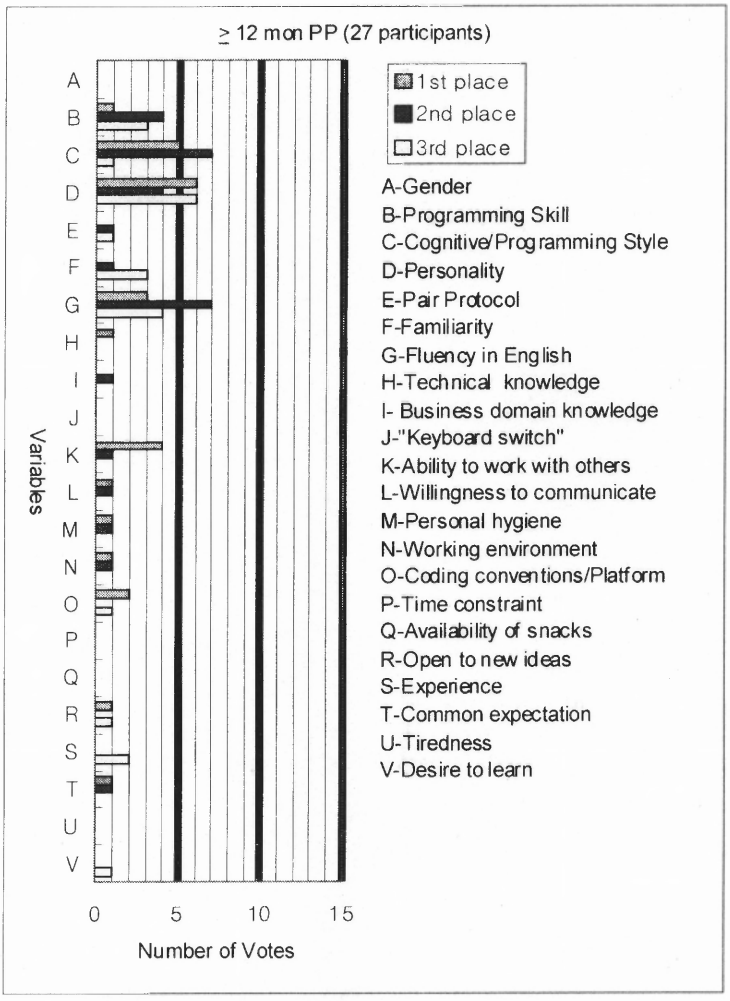


Figure 3.8 Ranking result from the subjects who possess greater or equal to 12 months of PP experience.

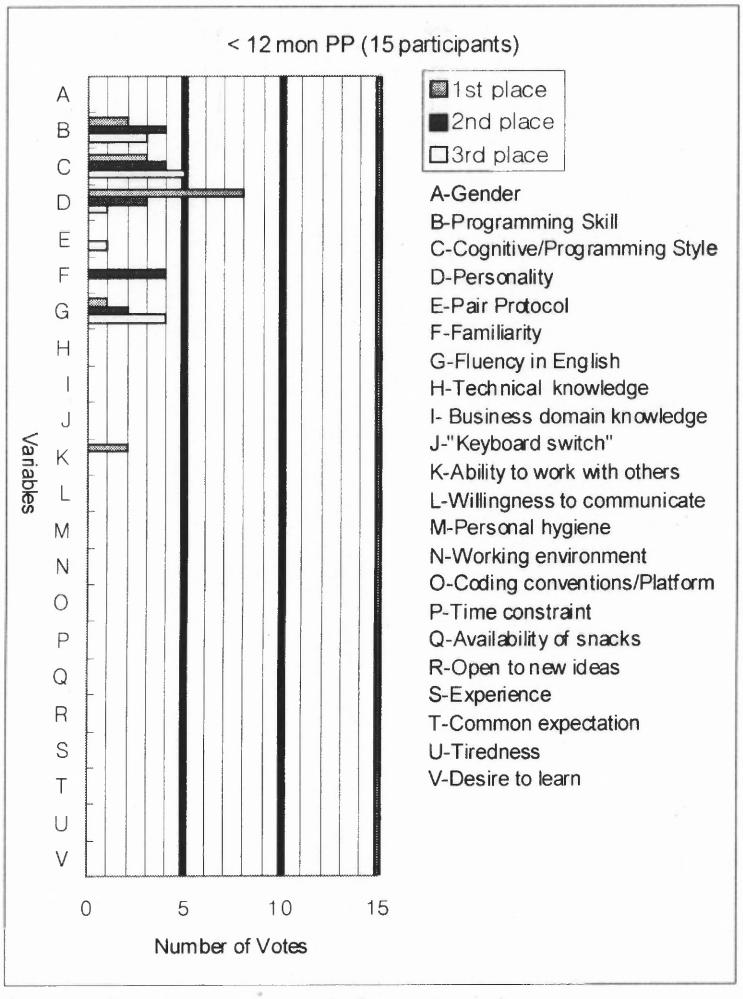


Figure 3.9 Ranking result from the subjects who possess less than 12 months of PP experience.

In "<12 mon. PP" group, the variable 'personality' outstands with its 1st place votes. Except 'ability to work with others,' no third place-and-up votes for the additional variables. This is a contrast to the "≥12 mon. PP" group where the votes are spread out throughout all variables. A cautious attempt in explaining the most first place votes for 'personality' in "<12 mon. PP" group would be the early learning curve of PP. The spontaneous and demanding PP requires one to quickly adjust to or be synchronized with the partner. In meeting this requirement, dealing with partner's incompatible personality can be a challenge in an early learning stage of PP.

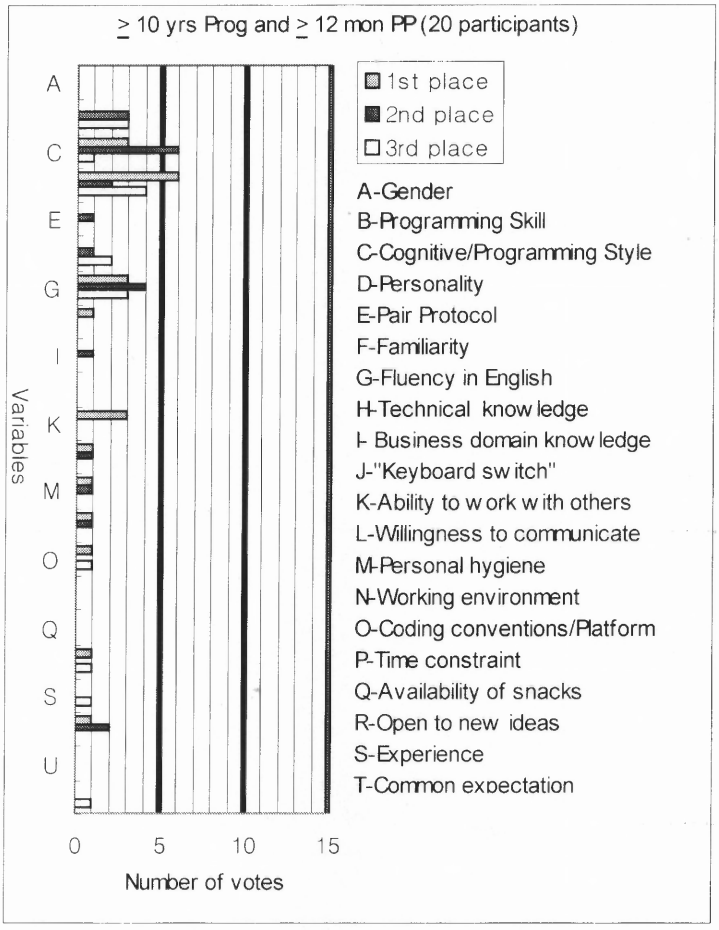


Figure 3.10 Ranking result from the subjects who possess greater or equal to 10 years programming and greater or equal to 12 months of PP.

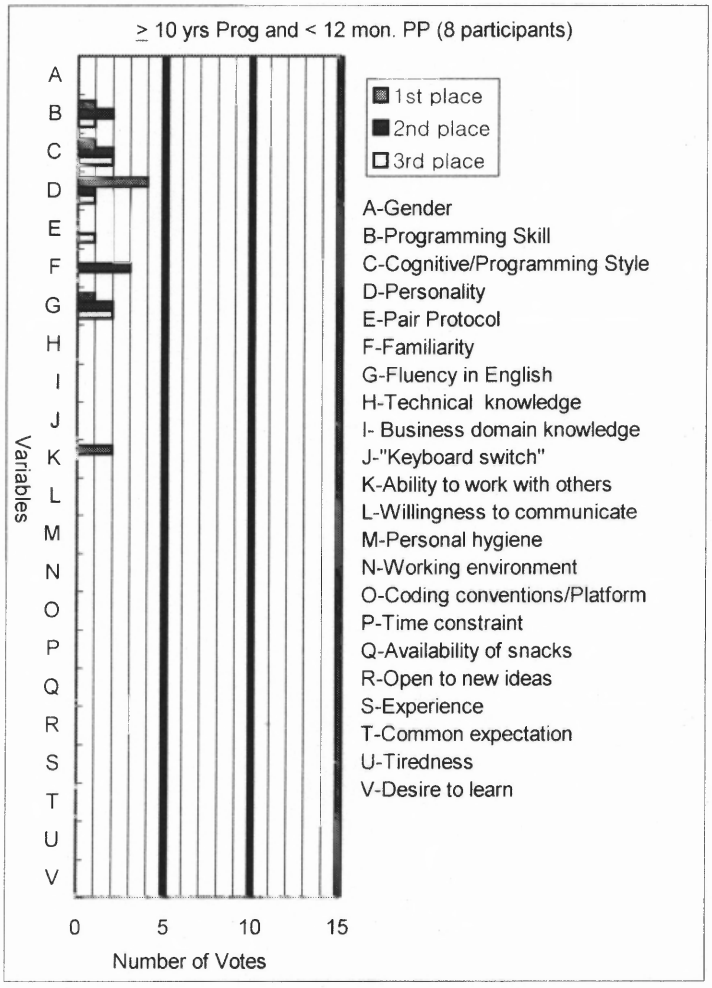


Figure 3.11 Ranking result from the subjects who possess greater or equal to 10 years programming and less than 12 months of PP experience.

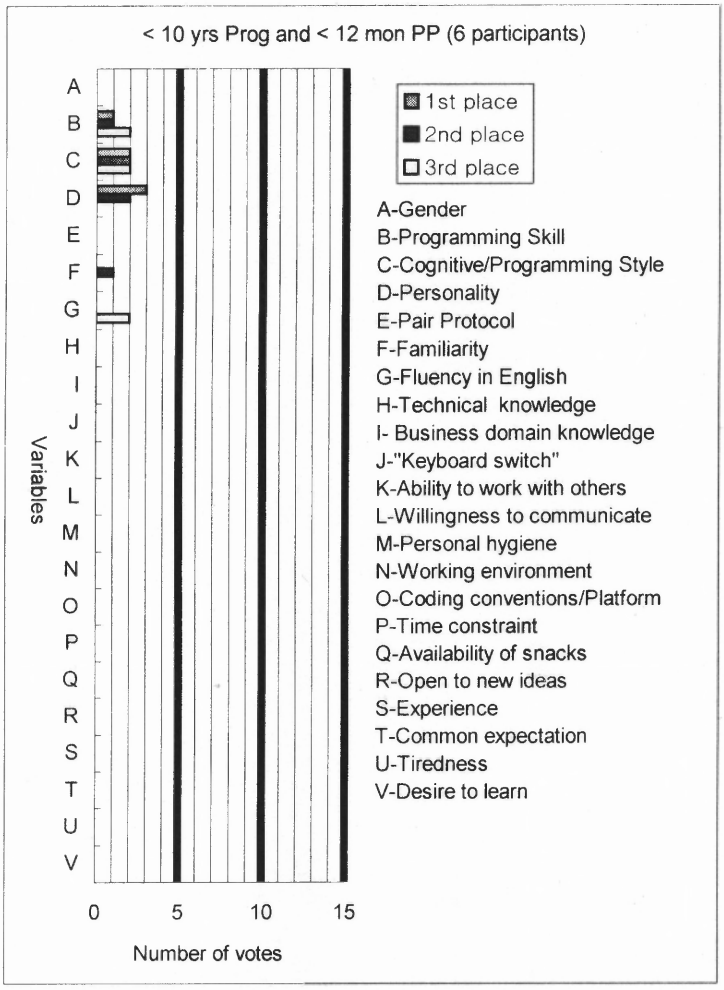


Figure 3.12 Ranking result from the subjects who possess less than 10 years programming and less than 12 months of PP experience.

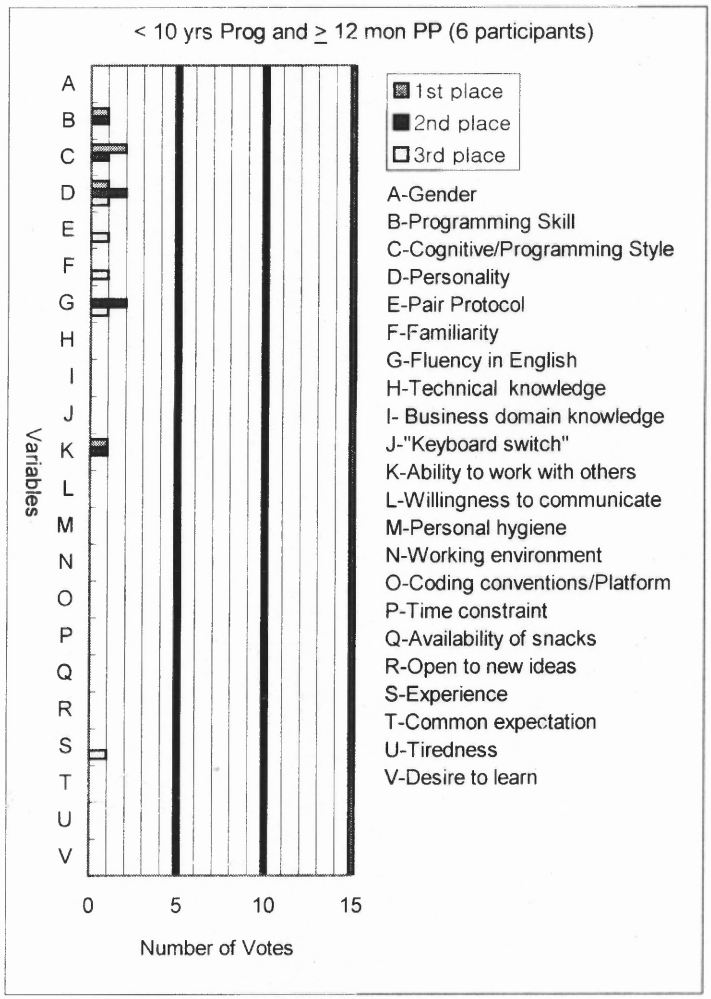


Figure 3.13 Ranking result from the subjects who possess less than 10 years programming and greater and equal than 12 months of PP.

Opposite to the anticipation, in the four groups, no notable relations of association are observed. The results of all four subgroups more or less have replicated the results of the previous four groups. Going by the 1st place votes, 'personality' has dominated in all four groups, followed by 'cognitive/programming style,' 'fluency in English' and

'programming skill' variables. Although statistically can be questionable with only six participants, the "< 10 yrs prog. and ≥ 12 mon. PP" has one 1st place vote for the 'personality' where it received three 1st place votes in "< 10 yrs prog and < 12 mon PP." In "≥ 10 yrs prog and ≥ 12 mon. PP" group, 'programming skill' did not receive any 1st place vote where in all other groups, it did with at least one 1st place vote each. One interesting observation is that the generally more experienced group values 'fluency in English,' the communication factor ahead of 'programming skill' and it is the opposite with less experienced group.

A comparison of "management" versus "non-management" was added in an effort to observe, if any, a difference in the perspectives of both. Here, the "management" is defined purely by the job titles such as Vice President, Senior Director, and Chief Technical Officer, and "non-management" with titles such as software engineer, software developers, consultant, and team partner. However, all have indicated that they themselves have either practiced or currently are practicing PP.

All groups more or less have replicated the summary result. One interesting observation is that the "more experienced" professionals value 'fluency in English,' the communication factor, ahead of 'programming skill' and it's the opposite with the "less experienced" professionals.

Table 3.2 Survey Result Statistic Figures

B = Programming Skill
C = Cognitive/Programming Style
D = Personality
E = Pair Protocol
F = Familiarity
G = Fluency in English (Communication)

Groups		B	C	D	E	F	G	Resp-onses
Professional Programming Experience	≥ 10 yrs.	3.90	3.43	2.92	6.18	5.20	3.28	28
	< 10 yrs.	4.08	3.29	2.21	6.08	5.15	4.43	14
Differences		-0.18	0.14	0.71	0.10	0.05	-1.15*	
Pair Programming Experience	≥ 12 mon.	4.50	3.74	3.13	6.55	5.58	3.38	27
	< 12 mon.	2.92	2.79	1.86	5.46	4.50	4.20	15
Differences		1.58**	0.95	1.27*	1.09	1.08	-0.82	
Management		3.00	3.13	2.14	5.00	4.63	2.75	9
Non-management		4.26	3.45	2.77	6.38	5.33	3.94	33
Differences		-1.26	-0.32	-0.63	-1.38*	-0.70	-1.19	

Significant at $\alpha = 0.1$ level ** : significant at $\alpha = 0.05$ level
(used a non-parametric test, Mann-Whitney difference test)

Table 3.2 displays the average ranking of each variable in its respective group. For example, 3.90 under B column means 'programming skill' is ranked 3.90 in the group who have ten or more years of professional programming experience. Under responses column, 28 means there are 28 persons in the group who have ten or more years of professional programming experience. The 'differences' row indicates the difference between the two above values, i.e. –0.18 = 3.90 – 4.08. Only B, C, D, E, F, and G variables are considered here because all variables after G do not carry enough data points to be considered. A, 'gender' variable, is omitted because it has consistently received the "rock bottom" lowest vote in all conditions. In the PP experience group (12 mon. PP) comparison, almost all average rank differences are one full rank different. B, 'programming skill', shows the largest difference with (1.58) which explains the

difference of the perspective. A possible explanation is that for the "more experienced" professionals the programming skill is no longer an obstacle whereas for the "less experienced" professionals it is. In the programming years, the second largest difference is D, 'personality' (0.71). In a similar fashion, an ability or attitude in dealing with different personalities of different PP partners is a bigger load for the "less experienced" professionals than the "more experienced."

In the management versus non-management sectors, one to three head ratio of management to non-management is shown. E, 'Pair protocol', is the largest difference, followed by B, 'Programming skill', and G, 'Communication'. For 'Pair protocol', one can only estimate that the management prefers the mode of "control and maintain" with a set of pair conducting protocols, but with the ranks of (5.00) and (6.38), it's hard to argue for its importance.

Through these ranking comparisons one ascertains the perspective differences and that the differences are mainly stemming from the group's characteristic. It is also observed that many psychosocial variables do play a rather large role in influencing PP productivity.

3.3.3 The Survey Participants' Comments

Besides the ranking, another rich piece of information is the participants' comments. Many comments illustrate the frontline PP experiences with one's subjective words. It has "color painted" the survey result. The comment section is a place where the participants can "add" to their variable choices and also express their personal views towards PP in general or towards this survey. The comments are very informative, insightful, and most of all very personal.

After voting 'personality' the number one variable, many went on to give more detailed comments as a support of the choice.

"Personality types are extremely important." Kent Beck (Beck, 2000) *is describing, "Watching another person program is like watching grass die in a desert" in Extreme Programming Explained. "Both programmers must be 'driving'."*

"Personality conflicts seem to be the biggest issue I've encountered during my experiences, and can really hinder productivity."

"The most impact was personality. We had one guy who did not really work well with others, did not like others changing HIS code, and did not want 'the spec' (user stories) to change!."

"The worst thing that can happen is when you alert your partner to a mistake and he starts to defend himself."

As expected, many comments emphasized the humanistic dimension of the pair interaction; "open to new ideas," "desire to learn," "ability to work with others," and "willingness to communicate" were echoed.

"The key parameter is to have an open mind. I have found from experience that working with beginners can be as enriching as working with experienced persons. Listening to the other person, and respecting him/her are the only two ingredient for success".

"Pair prog. takes patience and a desire to make it work on the part of both parties. The faster more facile one must realize that he/she will make it farther with the help of the weaker programmer, and be able to see the advantage of that: "courage" ".

"Pairs have to understand each other and communicate well. They have to be able to "get along" with each other, but they don't have to be "buddies" or have the same culture".

"I think the most important thing to do with the productivity of the pairs would have to be each pairs willingness and ability to compromise and grasp concepts that the other member of the pair is espousing. Else, there is no point".

*"I have worked with some real ******** who think they know everything and pairing means "do it my way". I have worked with some real great people where I have learned a lot from them and I know they have learned from me".*

"Pair that listens to one another and are willing to try new ideas seems to achieve more. Explaining things to one another (without being patronizing) when one member of the pair doesn't understand a concept, is also helpful in increasing knowledge for the team".

"If the team has respect for each other, and enjoy working with each other they will do well. Protocols, Gender and Skills will be completely eclipsed by

a willingness to share ideas, and a desire to have conversations in voice and code at the same time".

They may have expressed the views differently, but they are collectively voicing teamwork and openness, which appear to be significant more so in PP environment than an ordinary software development team. There were also a number of comments that were convinced of the effectiveness of XP and PP. Most mentioned the code quality and the knowledge transfer as the common benefits. Reducing the tedious code debugging and the knowledge transferring between the senior programmers and junior programmers without setting aside additional training time was also praised.

"The one thing that needs to be present is an open-mindedness and a willingness to try PP the first time. After that, I think anyone who gives it an honest try can see the real value in it, and truly believes that it is better (i.e. more productive, more fun, more rewarding, and results in higher quality designs) than 2 people working separately".

"Extreme programming is great, increases productivity and especially quality, distributes knowledge among team".

"We had a mix of skill levels (Lisp novice to very-skilled practitioners). In our experience the weak got strong and the strong got stronger. We put out the highest code quality and had the highest programmer productivity I have seen in 20 years of software development".

"If the time is managed well, with sufficient breaks, pair programming can be very successful. The software product is at a much higher quality than it

would have been because of the peer pressure. The timesaving isn't half as long as a single programmer but maybe 75% of what a single programmer would do. The biggest benefit is pairing junior programmers with senior programmer and getting the transfer of knowledge. It doesn't slow the senior down very much, but the junior gets a huge boost in understanding the problem area and programming knowledge. It reduces the time for a new programmer to be a contributor to a project".

As most responses were favorable of XP and PP, but in all fairness these comments are from likely XP proponents, coming from XP favored discussion groups. Maybe the most challenging part of PP is the "introduction." Breaking the stereotype and dealing with one's stubbornness of insisting one-man programming are truly a tough task for any manager. Whatever the reasons are, the introduction of PP appears to be the most stifling step in the adopting process.

"One of the critical issues with pair programming is getting people to actually start doing it. After some time, people will value it and use, but getting them to this point is the problem I'm still struggling with this step. It might be interesting to also investigate the factors to better facilitate initial adoption of pair programming".

For 'gender' issue, it was found to be insignificant from this survey. The following comments typify the consensus;

"I really don't have any experience pair programming with a female, but I don't know think gender should matter".

A female professional, has commented;

"Note I am female working in a team of males I think that this is sometimes an issue when pairing".

Unfortunately, no conclusive data is available at this time to elaborate in gender issue, but with the last place finishing in all conditions, 'gender' appears to be the least influencing variable in PP productivity.

Even with the "40 hrs a week/ 8hrs a day" schedule (Beck, 2000), the condition of PP constantly challenges the programmers. It appears that there are physical and psychological issues. Physically, the programmers face the mental fatigue or the "burn out." The mental fatigue may lead to a possible deterioration in code quality and lowers productivity as well as the pair's quality relationship.

"Since people cannot effectively work at such high-intensity for extended periods without break, it is essential that the human needs of the pair-team individual also be paid 'extreme' attention".

"Pair programming can be very tiring over extended periods of time. It is very difficult to pair program 8 hours a day 5 days a week. It is very intense. When two people are

concentrated working on a solution there are a lot of ideas and discussion that require a lot of mental energy. You program at a faster pace when pairing and don't take breaks when you normally would since you're normally involved programming or discussing something while you pair. You have to force yourself to take regular breaks so that you don't burn out. You can tell when you have been pairing too long because you become impatient with one another, argue more, and feel extremely tired."

Psychologically, the programmers deal with personal favoritism, preferences, and politics. Although this is thought to be minimal, the survey result indicates it's more than minimal. Today's current practice of PP is "simply pair up" without any pre-consideration. A worse case scenario is the result of low morale as the following comment shows:

"*If the selection of an extreme programming partner for the expert is based on his or her personal preference, then this may demoralize other members in the group. Why should a person be chosen for learning an expertise not because of the ability to learn and the desire to learn, but rather because of the personal prejudice of the partner."*

Given these issues, the role of management is important here. These physical and psychological concerns can only be alleviated by the management's assertive

involvement. Instead of saying "team up with one who you want to work with," the management needs to direct and "fairly officiate" the process. There are projects that can be done with teams of "buddies", but then there are projects that are not. Most management looks for the project's importance, size, time requirement, the programmers' programming skills, and other explicit characteristics. But the smart management will also consider these physical and psychological issues and deal with them appropriately.

Through the survey, the factors that the professionals think impact PP productivity have been revealed. The result tells that it's the humanistic variables over technical variables that are viewed more significantly by the professionals. Lead by 'personality,' psychosocial aspect appears to be more than a mere factor and merits more research.

3.4 Hypotheses Development

Based on the survey result, one may now understand what factors are being considered, at least from a group of very experienced programmers and management. Given the non-existence of prior psychosocial PP studies, this "opinions" or personal accounts may be a good hint to start a study. The psychosocial fact has shown that it is far more important in PP than one-man programming.

In one-man programming, the psychosocial term implies to group collaboration or teamwork that consist of two or more programmers (Weinberg, 1998). But the two-programmers team is mutual programming, not pair programming. Here, mutual programming can be defined as two programmers collaborate in an assignment by

programming individually. Physically they are not sitting "lap-to-lap" like pair programming.

Cognitively, this is completely different than in pair programming. It's like a two-headed monster, one body but two heads, one PC, monitor, keyboard, but two programmers. Maybe this is why 'personality,' 'communication,' 'cognitive/programming style,' 'ability to work with other' and other many psychosocial variables were surfaced from the field survey. Looking at the top choices, among others 'personality' and 'communication' the 1st and 3rd choices, present very old and familiar topics in social psychology. But in PP context, they are new topics. For the success of PP and its future, it just maybe that how active research of these topics would decide that. The difficulties of lack of programming skill and other technical know-how can be overcome administratively, but the difficulties of incompatible or "mad-bomber" (Weinberg, 1998) like personalities may have no solution.

In this experiment, the 'personality' and 'communication' factors were selected to study. However, other variables are equal in value for experiment as well. The task of reviewing two variables is certainly a tall order and this is not a place for it, but an attempt of a basic review according to PP experiment objective is made.

3.4.1 Personality Concept Review

In no comparison to the robust personality researches in social psychology, the experiments involving personality in programming context is very limited and in PP context, it's virtually non-existent. There are literatures that deal with psychology in systems, in a more general sense, (Weinberg, 1998; Shneiderman, 1998; Shaft and Vessey, 1998), but definitely no literatures with any PP specificity. In programming, one

generally believe that a typical programmer personality profile is "detached" and modifiable through "attempts at adaptation" (Weinberg, 1998). Yet, this is a contrary to some personality views in social psychology.

The two sides of personality, temperament and character where temperament is predisposition and character is disposition, tells that temperament is what a person is born with, it can't be changed (Keirsey, 1998). Regardless of changing environment, there is a part of individual that won't change. With this in mind, studies in knowing that predisposition continue. In early, there were attempts to profile programmer personality for a hiring purpose, but with a little success.

In social psychology, a typical personality theory may be described by using 3 to 7 dimensions (Revelle, website A; Pervin, 1989). Most common ones are 'Big 3' and 'Big 5' models. In 'Big 3' model (Revelle, website B), the 3 dimensions are 1) Approach - Instigation of Behavior, 2) Avoidance - Inhibition of Behavior, and 3) Aggression. The "detached" personality corresponds to avoidance, inhibition of behavior. Increasing the number of dimensions, today, 'Big 5' is better accepted and used model (Pervin, 1989). Big 5 has 1) Extraversion, 2) Agreeableness, 3) Conscientiousness, 4) Neuroticism, and 5) Openness.

Table 3.2 'Big 5' Concept Defined

Dimensions	*Characteristics*
Neuroticism	Proneness to psychological distress, excessive carvings or urges, unrealistic ideas
Extraversion	Capacity for joy, need for stimulation
Openness	Toleration for & exploration of the unfamiliar
Agreeableness	One's orientation along a continuum from compassion to antagonism in thoughts, feelings, and actions
Conscientiousness	Individual has degree of organization, persistence, and motivation in goal-directed behavior

In each dimension, there are two extreme ends: one end is the most positive side and the other is most negative side. On a scale, an individual with a higher score is said to have the positive end of the dimension. Following figure illustrates both negative and positive sides of all five dimensions.

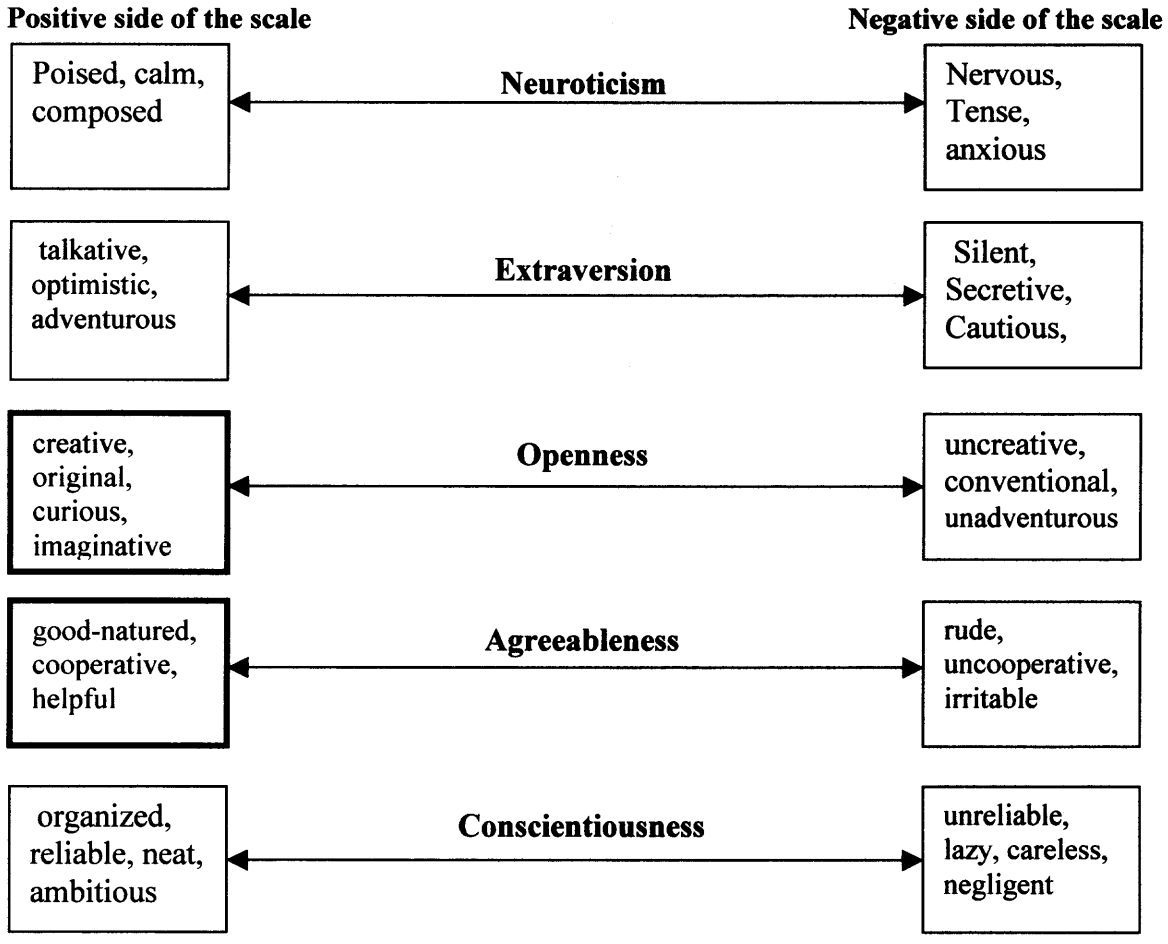


Figure 3.14 'Big 5' illustrated.

According to the 'personality' variables and other relevant qualities (or the variables) that were obtained from the survey, they indicate a closer link to the positive sides of 'Openness' and 'Agreeableness.'

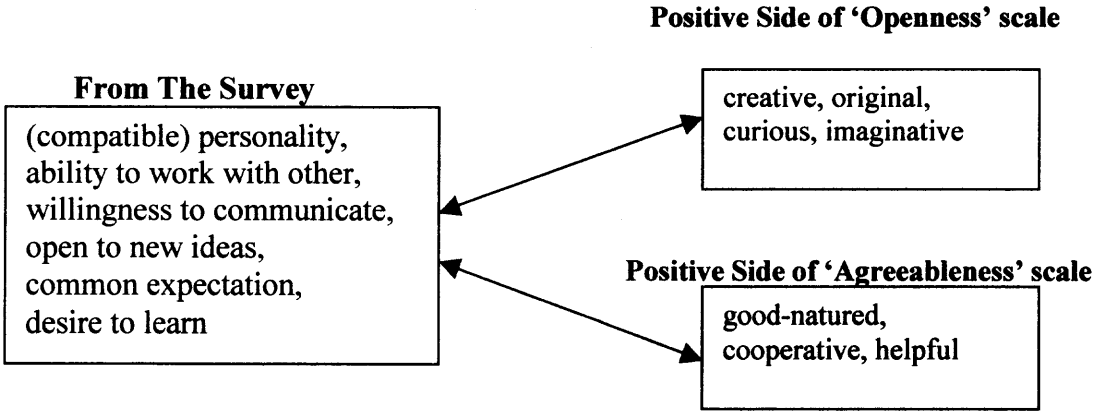


Figure 3.15 Linkage to 'Openness' and 'Agreeableness' scales.

Debatable, but reasonable, one can make the careful ties between the survey variables and the positive sides of 'Openness,' and 'Agreeableness' scales. In summary, one may say that a person who scores high on the scale, possessing positive qualities of 'Openness,' and 'Agreeableness' is the likely the person who fits the description of the list from the survey. Following quote puts it well:

> "*When I ask clients about their best experiences of being part of a team, the most often cited memory is that it was like a family-a family around Thanksgiving time. Everybody makes something and brings it to the table, where they can all share and celebrate. I think another attribute of a healthy team is that it can perpetuate itself. It can help a member break off and start and support a new team that will share the same values and customs.*" (Weinberg, 1998)

The next task is selecting appropriate personality theory for the experiment. Following 'Big 5' Table shows a list of personality theories described in its five dimensions.

Table 3.3 Personality Theories

Theorist/ Inventory	I Extraversion	II Agreeableness	III Conscientiousness	IV Neuroticism	V Openness
Bales	Dominant-Initiative	Social-Emotional Orientation	Task Orientation		
Block	Low Ego Control		High Ego Control	Ego Resiliency	
Buss & Plomin EASI	Activity Sociability		Impulsivity (r)	Emotionality (r)	
Cattell 16PF	Exvia (vs. Invia)	Pathemia (vs. Cortertia)	Super Ego Strength	Adjustment vs. Anxiety	Independence vs. Subduedness
Comrey CPS	Extraversion and Activity	Femininity	Orderliness and Social Conformity	Emotional Stability	Rebelliousness
Costa & McCrae NEO-PI	Extraversion	Agreeableness	Conscientiousness	Neuroticsm (r)	Openness
Eysenck EPQ	Extraversion	Psychoticism (r)	Neuroticism (r)		
Goldberg FFI	Extraversion	Agreeableness	Conscientiousness	Emotional Stability	Openness
Gough CPI Factors	Extraversion	Consensuality	Control		Flexibility
Guilford	Social Activity	Paranoid Disposition (r)	Thinking Introversion	Emotional Stability	
Hogan HPI	Ambition and Sociability	Likeability	Prudence	Adjustment	Intellectance
Jackson PRF	Outgoing, Social Leadership	Self Protective Orientation (r)	Work Orientation	Dependence (r)	Aesthetic-Intellectual
Myers-Briggs	**Extraversion vs. Introversion**	**Feeling vs. Thinking**	**Judging vs. Perception**		**Intuition vs. Sensing**
Tellegen MPQ	Positive Emotionality	Constraint	Negative Emotionality	Absorption	
Wiggins IAS	Power/ Dominance	Love/ Warmth			
Zuckerman	Extraversion		Psychoticism/ Impulsivity/ Sensation Seeking (r)	Neuroticism (r)	P-Imp-SS

In the table 3.3, each personality inventory has been carefully analyzed against the five dimensions and labeled accordingly. However, one must caution in interpreting the table; it would be risky to assume that a particular dimension of a personality theory has an exact "one-to-one" relationship with the corresponding dimension of the big five dimensions. A more appropriate way to address this is that the particular characteristic of the personality is a "closer" match to that dimension.

Reviewing the Table, one that is most widely used and familiar to public is focused on. One that fits the billing is Myers-Briggs Type Indicator (MTBI) (Bayne, 1995; Keirsey, 1998; Myers & Myers, 1995) as it's used in 84 of the Fortune 100 companies and more than 50 different countries according to CCP Inc., the official MBTI material distributor (CCP website 1). According to the five dimensions of 'Big 5,' MTBI fit into four of five dimensions.

Table 3.4 Myers-Briggs Type Indicator (MBTI) from 'Big 5'

Big 5 Dimensions	**Myers-Briggs Type Indicator (MTBI)**
Neuroticism	———
Extraversion	Extraversion (E) vs. Introversion (I)
Openness	Intuition (N) vs. Sensing (S)
Agreeableness	Feeling (F) vs. Thinking (T)
Conscientiousness	Judging (J) vs. Perception (P)

For 'Openness' and 'Agreeableness,' MBTI has 'Intuition vs. Sensing' and 'Feeling vs. Thinking.' 'Intuition vs. Sensing' - Simply put, sensing is where a person is paying attention to the stimulus of surrounding environments and intuition is where the person listens to "inner voice." Everyone has both tendencies, but one dominates other. Everyone tends to have one dominant character over the other. But at the same time, the two tendencies are not mutually exclusive; meaning a person with strong intuition tendency often observes his environment and vice versa with a person with sensing tendency. For sensing, the qualities are creative, original, curious, imaginative and more, for intuition, the qualities are unartistic, conventional and more. This measurement is structured so that a high score on this scale indicates sensing, and a low score indicates intuition. Therefore, if individual scored high on this scale means he has stronger tendency of sensing over intuition.

'Feeling vs. Thinking' – For feeling one may use words such as "tender-minded," "friendly" and for thinking one may use "tough-minded." All have thoughts and feelings but there are individuals who do more of one over the other. The common stereotype is that one is sensitive, warm-hearted, and cooperative and the other is cold-blooded and insensitive, but the truth is that both react in same intensity. The difference comes from how it's being displayed. The "tender-minded" is very expressive in revealing his or her feeling whereas the "tough-minded" completely hides his or her feelings. Hence, in summary, both react emotionally in same intensity, but the display of emotions is different. One may describe the feeling in words such as good-natured, trusting, and helpful, for the thinking one may use words such as uncooperative and irritable. This measurement is structured so that a high score on this scale indicates

feeling, and a low score indicates thinking. Therefore, if individual scored high on this scale means he has stronger tendency of feeling over thinking.

Table 3.5 Eight Preferences of MBTI Type Indicator

Where a person focuses his or her attention	**E**	EXTRAVERSION People who prefer Extraversion tend to focus on the outer world of people and things	**I**	INTROVERSION People who prefer introversion tend to focus on the inner world of ideas and impressions
The way a person gathers information	**S**	SENSING People who prefer sensing tend to focus on the present and on concrete information gained from their senses	**N**	INTUITION People who prefer intuition tend to focus on the future, with a view toward patterns and possibilities
The way a person makes decisions	**T**	THINKING People who prefer thinking tend to base their decisions primarily on logic and on objective analysis of cause and effect	**F**	FEELING People who prefer feeling tend to base their decisions primarily on values and on subjective evaluation of person-centered concerns
How a person deals with the outer world	**J**	JUDGING People who prefer judging tend to like a planned and organized approach to life and prefer to have things settled	**P**	PERCEIVING People who prefer perceiving tend to like a flexible and spontaneous approach to life and prefer to keep their options open

In MBTI, a person is given a four-letter type indication such as ENFP where each letter is coming from each of the four scales or preferences of MBTI. The E is from Extraversion (E) vs. Introversion (I) scale, N is from Intuition (N) vs. Sensing (S) scale, F is from Feeling (F) vs. Thinking (T) scale, and P is from Judging (J) vs. Perception (P). Except the ones who are in the field of psychology one typically know about MBTI up to this point, the four-letter type, but the 'type theory' reveals the true meaning of the four-letter type.

The type theory generally says that one character, type or preference of the four preferences (Sensing, Intuition, Thinking or Feeling) usually dominates the others. A person uses this dominant type the most and feels most comfortable; it is an essential part of the person at his or her best (Bayne, 1995). A person uses the dominant type in his or her daily life, enjoy it, and trust it more. Adding it to the dominant type, there is auxiliary or secondary type. An auxiliary type is the second dominant type that a person uses it most next to the dominant type. It complements and supplements the dominant type. Along with the dominant type, the auxiliary type is readily used and a person frequently shifts back and forth between the dominant and auxiliary type without one's realization. This leads to following possible combinations; ST, SF, NF, and NT.

After the dominant and auxiliary types, xNFx, only the outer preferences remains, ExxP. For the extraverts and introverts, "*the extraverts use their dominant function mostly in the external world, because by definition that is where they are most comfortable, and introverts use theirs mostly in their inner world, for the same reason*" (Bayne, 1995). Extraversion vs. Introversion indicates one's preference in displaying and directing his or her dominant and auxiliary types. Thus, contrary to what many have believed, extravert is not always loud or happy smiley person and introvert is not always quiet or withdrawn person.

For the perception and judgment, they are the attitudes that one uses in dealing with the outer world. Some are quick to judge things or some are slowly and carefully listen and evaluate things. Typically, a person starts with a set of particular preferences (Keirsey, 1998), but one can effort to, if he or she decides, develops a particular weak or less used preference, which is called 'type development' (Bayne, 1995; Myers & Myers, 1995). Therefore, it's clear now that a person is who he or she is by the dominant and

auxiliary types. This fact closely matches the list of survey results to the 'Openness' and 'Agreeableness' dimensions or intuition (N) vs. sensing (S) scale and feeling (F) vs. thinking (T) scale.

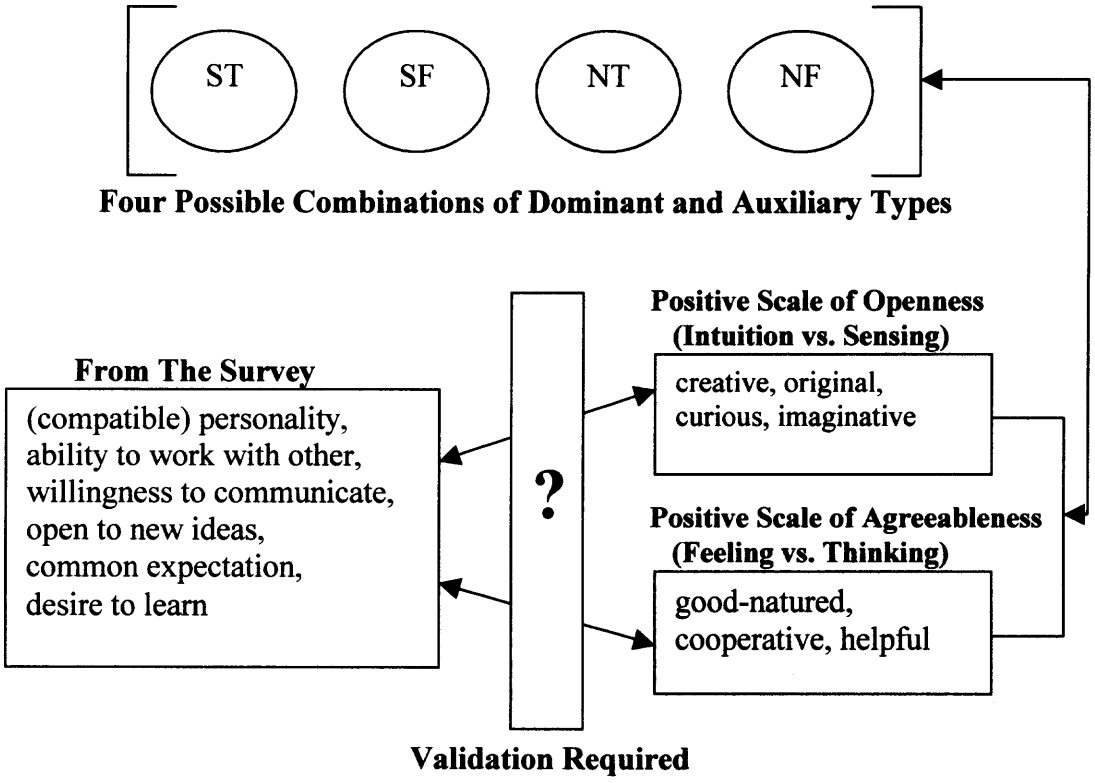


Figure 3.16 MBTI type theory to the survey variables.

Through the type theory, the link between the list of survey variables to the four possible combinations of dominant and auxiliary types substantiates a validation by an empirical work. To some, personality issue in programming may not all that significant, but in PP it just may mean everything just as this survey result indicates. It would not be an overstatement if one says that PP is more of a social activity than a programming

activity. The psychosocial dimension of programming is certainly not one of a top discussion topic, but now one understands its role and effect.

> *"Thus, all other things being equal, certain people will find the job of product test programmer easier psychologically."* (Weinberg, 1998)

3.4.2 Communication Concept Review

Increasingly, communication is valued as one of the core competencies in many organizations (O'Neil, et al., 1997). Communication is probably the most used word in teamwork and collaboration, most used because it's that important. In PP context, communication can be viewed as either very easy or very difficult because of "side-by-side," yet want to be "alone and be creative." Within communication theories and concepts (Spitzberg, 2002; Spitzberg 1997), efficient communication methods and evaluations are observed in various contexts. Communication is such a broad term that it's difficult to put it in one or two sentences. The communication or "exchange" can happen explicitly and implicitly, physically and psychologically, visually and audibly, or through other means of channels. Everyone communicates, but a communication skill may come as a vague and new item to some people. The 'skill' is defined as "*intentionally repeatable, goal-directed behaviors and behavior sequences*" (Spitzberg & Cupach, 1984). Anyone can communicate, but doing it in a manner that achieves a goal is considered a skill. This 'skill' concept is much more impact in PP context as demanding real-time non-stop cognitive pair collaboration is waits.

Just as the personalities are differ by the different combinations of dominant and auxiliary types, communication also appears to behave the same fashion (Thompson, 2000). In a similar manner communication also adheres to the type theory in shaping

what a person is most comfortable with. It's called "type dialects" and the basic idea is that a group of individuals with at least one common either dominant or auxiliary type speak their "own" language, it is subtle yet definite. For example, a ENFP person with N as the dominant and F as the auxiliary types communicates in a dialect with a person who possesses either N or F as a dominant or auxiliary types. This aligns with the type theory of personality. The relationship between the communication dialects of type theory and communication 'skill' is unknown. One does not know whether one's communication skill has any effect on the communication dialect or that a certain combination of dominant and auxiliary type possesses high level of communication 'skill.' It is one of the objectives of this experiment to investigate the effect.

3.4.3 Independent Variables

The three main independent variables are: 1) the dominant and auxiliary preferences of the MBTI type, 2) the communication skill level, and 3) gender.

Table 3.6 The Dominant and Auxiliary Preferences of The MBTI Type

ST	SF	NF	NT
ISTJ	ISFJ	INFJ	INTJ
ISTP	ISFP	INFP	INTP
ESTP	ESFP	ENFP	ENTP
ESTJ	ESFJ	ENFJ	ENTJ

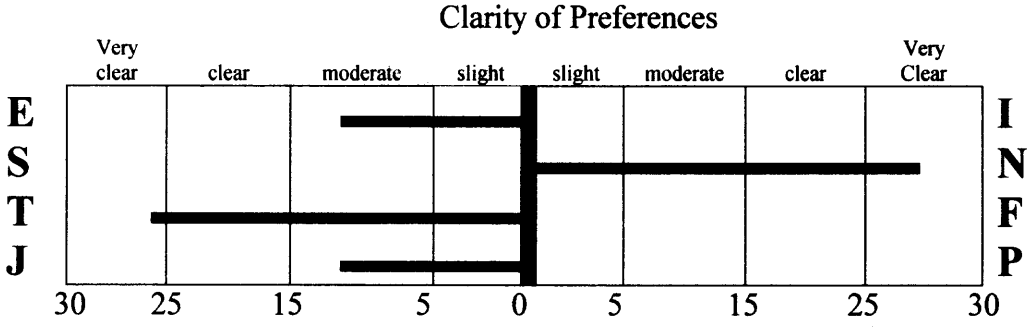


Figure 3.17 The preferences' strengths of MBTI type.

The bars on the graph below illustrate a person's MBTI preference clarity. The length of each bar shows how consistently the person chooses one pole of a preference over its opposite. A longer bar suggests that he is quite sure that he prefers that pole; a shorter bar suggests that he is less sure about his preference for that pole.

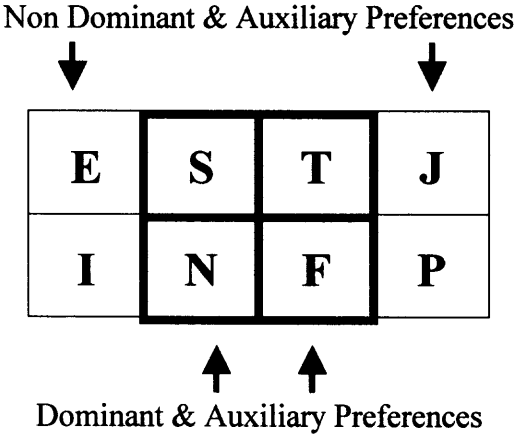


Figure 3.18 Illustration of non-dominant & auxiliary preferences.

In MBTI, there are all together 16 types (Table 18). As an independent variable, the focus is on just the dominant and auxiliary preferences, [sensing (S) vs. intuition (N)]

and [feeling (F) vs. thinking (T)], the two inner preferences. In Table 18, the possible combinations are ST, SF, NF, and NT and there are 16 MBTI types. From these one can derive the with following three distinct groups:

1) "Diverse" pair: This is a pair of subjects who are alike in EITHER their dominant OR auxiliary preferences but not both (ST-SF, NT-NF, ST-NT, or SF-NF).
2) "Alike" pair: This is a pair of subjects who are alike in BOTH their dominant AND auxiliary preferences but not both (ST-ST, NF-NF, NT-NT, or SF-SF).
3) "Opposite" pair: This is a pair of subjects who are completely OPPOSITE in BOTH their dominant auxiliary preferences (ST-NF or NT-SF).

These three groups are used as the independent variable.

The Communication Skill Level

In determining a subject's communication skill level, we have the process illustrated in the preparing experiment material section, and appendix E, F, and G. Qualified subject is labeled with either 'High', or 'Low' for his or her communication skill level. For the pairing, 'High' and 'Low' groups are paired to High-High (HH), High-Low (HL), and Low-Low (LL). These three groups are used as an independent variable.

Gender

For gender, three groups are formed: Male-Male, Male-Female, and Female-Female. These three groups are used as an independent variable.

Other variable consideration is the subject's class performance. With the subjects' consent, each subject's current programming class performance is obtained. To eliminate or minimize the subject's class performance, the subjects are to be paired according to their class performance.

3.4.4 Dependent Variables

Scoring is done in two categories; the code productivity and code design. Under the umbrella of the code design, the code efficiency and code readability are included as well. For the design and efficiency, one can choose structured, modular, or object-oriented approach, With regards to readability, perfect readability can be measured by a scenario in which a third person should not have any difficulty understanding and reproducing the code at another time.

The subjects were asked to print and submit hardcopies of everything that they have generated, even the error messages. Contextually, the code productivity consists of codes and code output that are produced by subjects. For the code design, depend on the nature of problems, one can choose structured, modular, or object-oriented programming technique. Also code efficiency and code readability are considered for code design. In summary, the code design is qualitative measurement whereas the code productivity is quantitative measurement. After complete review of the submitted printouts by subjects, the judges were instructed to give each score for code productivity and code design, from a scale of 0 to 10.

At the end of each visit. a post-session questionnaire is given and collected. Items interested are PP communication, PP confidence, PP compatibility, and PP satisfaction.

3.4.5 Hypotheses

In this experiment, the hypotheses are designed to test three areas. They are 1) optimal MBTI type pairing 2) communication skill and 3) gender pairing.

Optimal MBTI type pairing – This hypotheses is the main contribution of this experiment. Based on the survey result, 'personality' is voted as the most influential factor to PP productivity. From a productivity perspective, a mixed MBTI type will produce better. As one of the MBTI founders, Isbel Briggs Myers (1995) said, "*two people, alike in their kind of perception or their kind of judgment but not both, have the makings of a good working relationship. Their shared preferences gives them common ground and their dissimilar preference gives them, as a team, a wider range of expertness than either has alone*". 'Expertness' also infers to insight or inquiry. Two complementing programmers, each with its own unique perception (sensing vs. intuition scale) or judgment (feeling vs. thinking scale) would produce a higher output in the PP environment. As in all cases, diversity is always better. Also, for communication, a number of evidences (Thompson, 2000; Bayne, 1995; Myers & Myers, 1995) suggest that complete opposites in both perception and judgment will hinder communication.

H 1: The pairs who are alike in their kind of perception or judgment (dominant and auxiliary types only) but not both, [divrs], would achieve significantly higher **[score]** than the pairs who are alike in both, [alike]:

H 1.1: **[score]** – code productivity, P

H 1.2: **[score]** – code design, D

Diverse pairs are: [ST-SF], [NT-NF], [ST-NT], or [SF-NF].

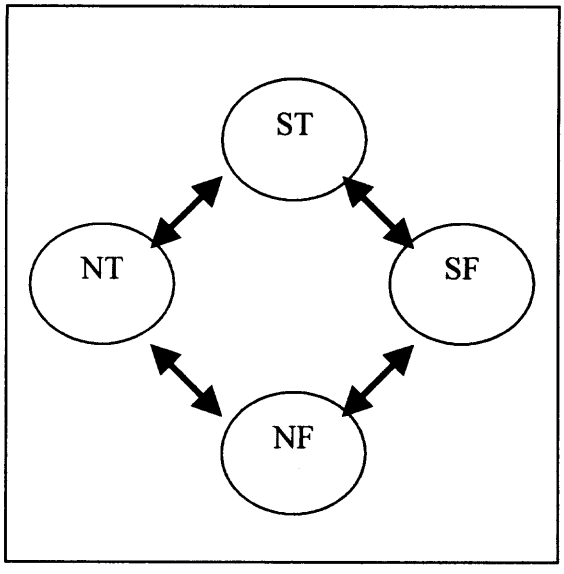


Figure 3.19 The diverse pairs.

H 2: The pairs who are alike in their kind of perception or judgment (dominant and auxiliary types only) but not both, [divrs], would achieve significantly higher **[score]** than the pairs who are opposite in both [opp]:

H 2.1: **[score]** – code productivity

H 2.2: **[score]** – code design

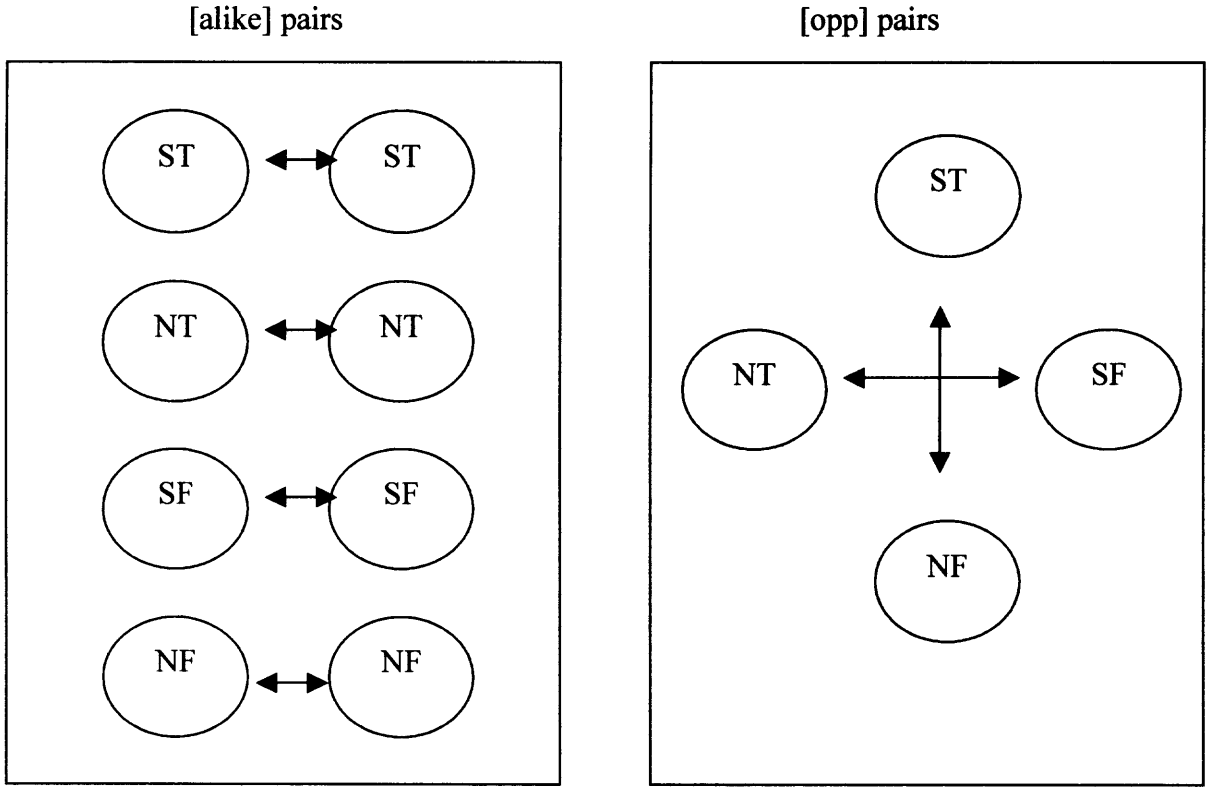


Figure 3.20 Illustration of [alike] and [opp] pairs.

H 3: The high communication skill level pairs, [HH], would yield a significantly higher **[score]** than the mixed communication skill level pairs, [HL].

H 3.1: **[score]** – code productivity

H 3.2: **[score]** – code design

H 4: The high communication skill level pairs, [HH], would yield a significantly higher **[score]** than the low communication skill level pairs, [LL].

H 4.1: **[score]** – code productivity

H 4.2: **[score]** – code design

Gender pairing – From the survey result and also the works from others (Williams, 2000; Williams & Kessler, 2002), no significant observations are made in the gender factor. However, the perception is that males and females are "different" cognitively and socially (Stewart, et al., 1996; Tannen, 1991), but complement each other in a team setting.

H 5: Male-Female pairs, [MF], would significantly achieve a higher **[score]** than Male- Male pairs, [MM].

H 5.1: **[score]** – code productivity

H 5.2: **[score]** – code design

H 6: Male-Female pairs, [MF], would significantly achieve a higher **[score]** than Female-Female pairs, [FF].

H 6.1: **[score]** – code productivity

H 6.2: **[score]** – code design

H 7: The pairs who are alike in their kind of perception or their kind of judgment (dominant and auxiliary types) but not both, [divrs], would achieve a significantly higher **level** than the pairs who are alike in both, [alike] from following constructs:

H 7.1: **level** - PP communication

H 7.2: **level** - PP satisfaction

H 7.3: **level** - PP confidence

H 7.4: **level** - PP compatibility

H 8: The pairs who are alike in their kind of perception or their kind of judgment (dominant and auxiliary types) but not both, [divrs], would achieve a significantly higher **level** than the pairs who are opposite in both, [opp], from following constructs:

H 8.1: **level** - PP communication

H 8.2: **level** - PP satisfaction

H 8.3: **level** - PP confidence

H 8.4: **level** - PP compatibility

H 9: The high communication skill pairs, [HH], would yield a significantly higher **level** than the mixed communication skill pairs, [HL], from following constructs:

H 9.1: **level** - PP communication

H 9.2: **level** - PP satisfaction

H 9.3: **level** - PP confidence

H 9.4: **level** - PP compatibility

H 10: The high communication skill pairs, [HH], would yield a significantly higher **level** than the low communication skill pairs, [LL], from following constructs:

H 10.1: **level** - PP communication

H 10.2: **level** - PP satisfaction

H 10.3: **level** - PP confidence

H 10.4: **level** - PP compatibility

H 11: Male-Female pairs, [MF], would significantly achieve a significantly higher **level** than Male-Male pairs, [MM], from following constructs:

H 11.1: **level** - PP communication

H 11.2: **level** - PP satisfaction

H 11.3: **level** - PP confidence

H 11.4: **level** - PP compatibility

H 12: Male-Female pairs, [MF], would significantly achieve a higher **level** than Female-Female pairs, [FF], from following constructs:

H 12.1: **level** - PP communication

H 12.2: **level** - PP satisfaction

H 12.3: **level** - PP confidence

H 12.4: **level** - PP compatibility

CHAPTER 4

EXPERIMENT

4.1 Experiment Design

4.1.1 Preparing Experiment Materials

[Experiment Location] – This experiment is to take place in a room located in the Guttenberg Information and Technology Center (GITC) Building, New Jersey Institute of Technology (NJIT) where noise and distractions are completely excluded or at least controllable.

[Participation of the subjects] – Inducing active or full participation from all qualified subjects is essential for a obtaining a reasonable sample size. A few visits to perspective subject classes were made to inform the subjects of the experiment's background and benefits. Moreover, in exchange for their participation, and with help from the course instructor, a course credit was given. The subjects' background and work experience were also reviewed for a consideration (Appendix B).

[Subject Consent Form] – A subject consent form is distributed which stipulates the rules and policies of the NJIT 'Protection for Human Subjects' committee (Appendix A). A detailed list of items and procedures are addressed as well as the subjects' rights during the experiment. Also, the notion of non-participating qualified subjects not being penalized in anyway is emphasized.

[Experiment Task] – After a review of the subject's programming experience, a set of four programming problems is designed by a panel of professional programmers, programming instructors and authors for the purpose of this experiment. The programming difficulty level of the four problems is set to challenge both undergraduate and graduate

subjects. And the parity of all problems is all equal as they have been reviewed and validated by three expert professional programmers (Appendix I).

[Communication Skill Measurement] – Measuring one's communication skill can be a difficult task. Hence, assessing one's communication skill is carefully approached and at the same time, accurately as possible. A few measuring instruments that had been used in previous research (Spitzberg, 2002; Spitzberg, 1997; Spitzberg and Cupach, 1984) was utilized in this experiment. During the first information session with the subjects, these instruments are used (Appendix E, F, and G). After a self-introduction, the pairs started the discussions on these topics. At the end, each was asked to fill out the Conversational Skills Rating Scale (CSRS) form (Spitzberg, 1997) (Appendix E) in which serves to evaluate the partner's communication skill. In labeling a person's communication skill level, the subjects were divided into percentiles, 40% and 60%, in order to create low ($\leq$ 40%) and high ($\geq$ 60%) competency groups.

[Personality Profile Measurement] – Myers-Briggs Type Indicator (MBTI) measurement was administered to the subjects. The site address is (http://www.skillsone.com/), and the Consulting Psychologists Press Inc. (CPP Inc.), the official MBTI distribution organization, operates it. The online version is selected over pencil and paper version, not only for the convenience but also for accuracy in scoring. In a case where a switch from the online version to the manual version is required, one may do so without any loss of test integrity and validity. According to CCP Inc., only the certified MBTI personnel can administer MBTI test. However, this requirement may be waived by CCP Inc. by having a PhD student researcher completing the qualification form and having it co-signed by the advising professor. In order to administer the manual version, the following materials must be ordered from CCP Inc. and used:

Table 4.1 MBTI Test Materials

[redacted]
Form M Item Booklets Reusable
Form M Non-Prepaid Answer Sheets for template scoring
Form M Non-Prepaid Scoring Templates for template scoring of Step II Prepaid Answer Sheets (to derive 4-letter type,)
Form M Report Forms for template scoring
MBTI® Manual (A)

[Fixed time for programming session] – During the pilot experiment, one of the parameters being evaluated was the assigned time for each problem. This is critical as the given time should not be too short or too long in conforming to the objective of this experiment. Based on the pilot experiment result and observation, 45 minutes for each problem are allocated.

[Judges] – Two qualified independent programmers are asked to participate as the judges. One judge is an Information Systems department graduate student, who is also a teaching assistant to a programming course and the other judge is a programmer who holds B. S. in computer engineering, and B. S. in Computer Science. Both judges

and the authors have discussed about the experiment and the programming problem set (Appendix H). The judges were given the scoring instruction (Appendix J) and received full explanations for any of their questions. The judges reviewed and graded the subjects' work in two categories: 1) quantitative measurement (code productivity) and 2) qualitative measurement (code design). The grading scales are 0 to 10 with 0 being the lowest score and 10 being the highest score. Also, if there is a difference in an item's scoring of more than 1 between the two judges then they are asked to reconcile for adjustment.

4.1.2 Experiment Procedures – First Part

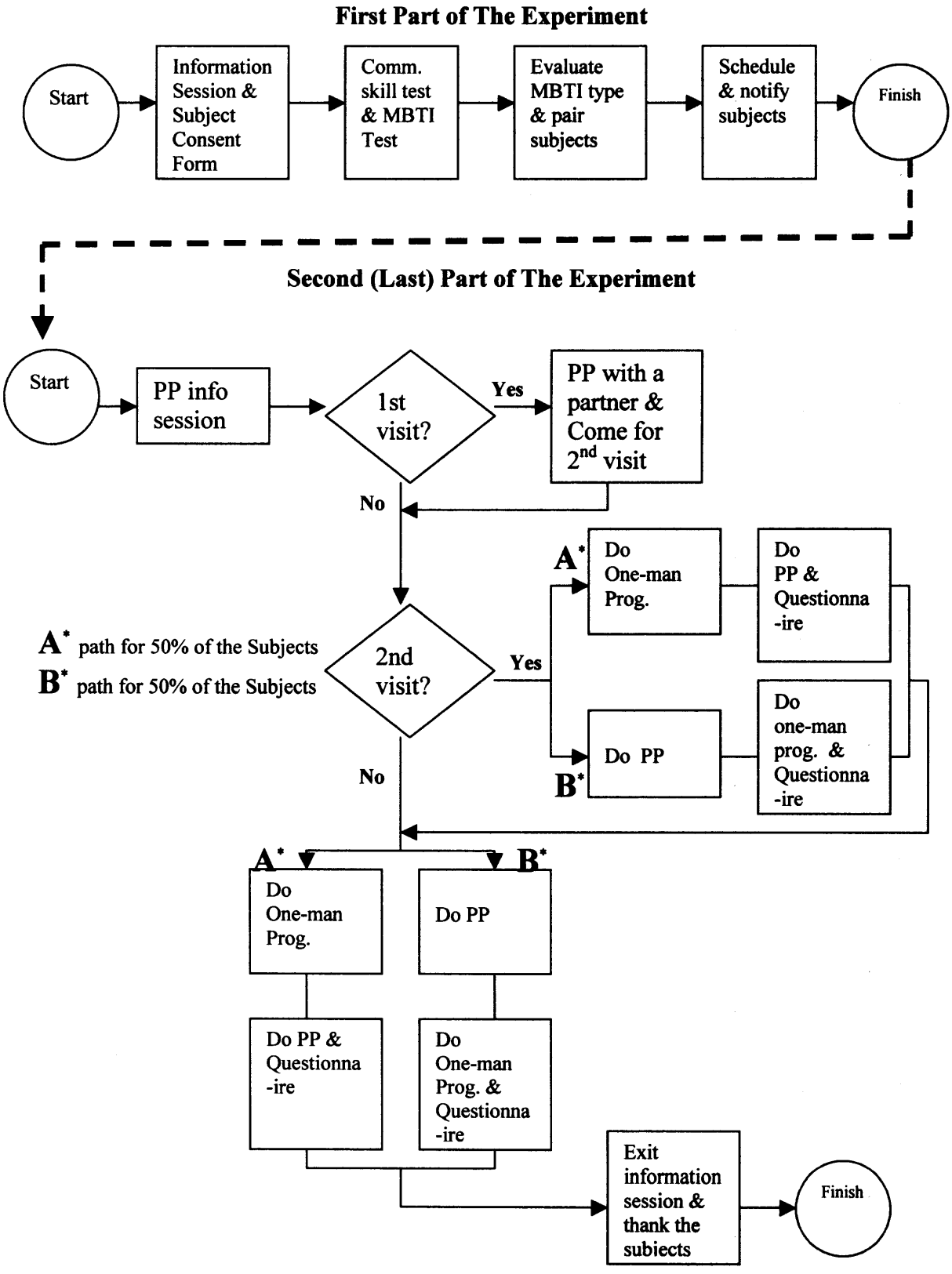


Figure 4.1 Experiment process flow.

The experiment largely consists of two parts. The first part consists of 1) holding an information/training session, 2) completing the subject consent form, 3) completing online MBTI profile measurement, 4) evaluating the subjects' profiles for pairing 5) scheduling the programming session dates and lastly 6) notifying the dates to the corresponding subjects.

[Information Session & Subject Consent Form] - Subjects will be asked to attend an information session and sign the subject consent form. The investigator gives an overall but brief background of the experiment and other PP research efforts. In verbatim, the subject consent form is read before the subjects and answers any concerns and questions from the subjects. The subjects are reminded of their freedom to withdraw from the experiment at anytime. At the end of session, he or she signs the consent form. This is also a training session. As has been documented in previous university experiments (Kivi, et al., 2000; Mueller and Tichy, 2001), the existence of no prior XP or PP training or knowledge is very risky to the experiment. The subjects in both experiments did not fully understand nor experience the true value and benefits of PP. However, realistically no subject can immediately turn into an experienced pair programmer after just one training session, but it does merit its place in this context. During the training session, the subjects are coached and lectured on the basics of PP (Williams and Kessler, 2000).

[Online MBTI personality profile measurement] – The subjects are introduced on MBTI background and its use. After a round of questions and answers, each subject is seated at a terminal and asked to complete the measurement. The analysis and profile result is executed by the site provider (http://www.skillsone.com/), the Consulting Psychologists Press Inc. (CPP Inc.), the official MBTI material distributor.

[Evaluating the subjects' profiles for pairing] – For the evaluation, the following data are to be assembled: course number, class performance, gender, programming experience, MBTI profile, the strength of each four preferences of MBTI, and communication skill result. The work experience of the subjects was carefully evaluated. A list of questions was presented to the subjects in order to obtain the subjects' programming backgrounds (Appendix B). On average, the professional programming experience is less than a half-year. For XP and PP experiences, only three subjects have stated that they have tried XP or PP for a brief period of time. Based on these results, it would be tenable to say that the programming experience of the subjects is minimal to none. Hence, the work experience of the subjects would not be considered as a factor in this experiment. According to the hypotheses and along with these data, all subjects are paired.

[Scheduling the programming session dates & notifying the corresponding subjects of the dates] – Barring any unforeseeable emergency, a total of three visits with the subjects is expected. In the real world, a software development effort goes on for months or sometimes for years. In this experiment context, three visits are made instead of one visit in an effort to more closely resemble a regular software project development effort. A semester long option was considered, however because of the large sample size and several administrative reasons, this was not feasible in this experiment. Once the pairs are identified, the three scheduled sessions are arranged, preferably in three consecutive weeks, one session per week.

4.1.3 Experiment Procedure - Second Part

The second or last part consists of the following steps:

- Two visits: each visit consists of a 90 minute programming session, 45 minutes in which the subject programs alone and another 45 minutes in which the subject programs in a PP environment.
- The order of programming alone and PP is alternated in two visits. Also, 50% of the sample pool starts with programming alone and the other 50% starts with PP.
- In the last visit, the subjects are provided with the information that had been hidden from them. Lastly, they are thanked for their participation.

[Programming Alone] – Each subject is given one of the four problems for programming alone. The subjects are instructed of the 45-minute time limit and that they are to turn in hard copies of their work at the end.

[Pair Programming] – The pair of assigned partners is given another problem out of the four problems that neither partner has done. Again, the subjects are told that 45 minutes are assigned and at the end that they are to turn in hard copies of their work. At the end of each visit, a post-session questionnaire is given to subjects.

[Exit information session & thank the subjects] – At the end of last visit, after the subjects have satisfactorily completed the entire task, they are given the information that they were being experimented on such as what type of MBTI group that they were in, which group of communication skill level that they were in, and what were the things that this experiment is observing. Also, any questions from the subjects are to be answered and a final thanks is given for their participation.

4.2 Experiment Result

4.2.1 Subjects Profile

For the experiment, the sample population is drawn from students in four programming courses: two on-campus undergraduate courses, and one online graduate course, and one on-campus course. However, all participating subjects have completed the experiment on campus. The students were given a course credit for their participation. The number of subjects represents those who have completed the experiment. . Except for the one course, the female students were scarce, thus the low participating female subjects.

Table 4.2 Subjects Profile – Course

Subjects	Undergraduate Course I (On-campus)	Undergraduate Course II (On-campus)	Graduate Course I (Online)	Graduate Course II (On-campus)	Total
Female Subjects	3	4	3	26	35
Male Subjects	37	24	3	28	93
Total	40	28	6	54	**128**

The MBTI types of all participated subjects (Table. 21) illustrate that most common type in computer field are ISTJ and ESTJ, in other words, people whose dominant and auxiliary preferences are sensing (S) and thinking (T) outnumber other combinations. This also confirms similar finding in other studies (Capretz, 2003). In the graduate course II, which may most resemble close to a real world with many working professionals or with prior experiences, ISTJ was most common. Introvert (I) preference appears to be common which is also a stereotypical in computer field (Weinberg, 1998).

Table 4.3 MBTI Type Distribution

Type	Undergraduate Course I	Undergraduate Course II	Graduate Course I	Graduate Course II	Total
ENFJ	2	1	0	0	3
ENFP	3	2	0	2	7
ENTJ	1	1	0	2	4
ENTP	0	1	0	3	4
ESFJ	2	2	0	3	7
ESFP	1	3	0	3	7
ESTJ	5	5	2	7	19
ESTP	5	2	0	0	7
INFJ	0	0	1	0	1
INFP	7	0	1	1	9
INTJ	1	2	0	2	5
INTP	3	1	0	5	9
ISFJ	1	2	0	4	7
ISFP	3	1	0	4	8
ISTJ	6	5	2	14	27
ISTP	0	0	0	4	4
Total	40	28	6	54	128

Table 4.4 Communication Skill Level Distribution

	Undergraduate Course I	Undergraduate Course II	Graduate Course I	Graduate Course II	Total
High	5	7	3	15	30
Medium	24	15	1	15	55
Low	11	4	2	21	38
Total	40	26	6	51	123*

* By an administration error, no data is available from five subjects

After the first visit, the information/training session, the data were juxtaposed and evaluated them for pairing. Upon reviewing the data and the hypotheses, following information was gathered for pairing. Where possible, even distribution was observed for all categories.

Table 4.5 MBTI Type Pairing

	Undergraduate Course I	Undergraduate Course II	Graduate Course I	Graduate Course II	Total
Diverse	7	4	0	10	21
Alike	7	6	1	9	23
Opposite	6	4	2	8	20
Total	20	14	3	27	**64**

Table 4.6 Communication Skill Pairing

	Undergraduate Course I	Undergraduate Course II	Graduate Course I	Graduate Course II	Total
H-H	2	2	2	1	7
H-L	8	2	0	13	23
L-L	4	1	0	2	7
*N/A	6	9	1	11	27
Total	20	14	3	27	**64**

* These pairs are the subjects who did not participate in comm. skill measure and also the subjects who are 'medium' comm. skill level.

Table 4.7 Gender Pairing

	Undergraduate Course I	Undergraduate Course II	Graduate Course I	Graduate Course II	Total
Male-Male	17	10	1	7	35
Male-Female	3	4	2	14	23
Female-Female	0	0	0	6	6
Total	20	14	3	27	**64**

4.2.2 Factor Analysis and Reliability Analysis

Generally, the emphasis of validating instruments in IS research is not publicized enough (Boudreau, et al., 2001; Straub, 1989). In fact, the instrument validation is always hidden behind the spotlight of research findings. By validating the instruments that one can substantiate the findings. In this experiment, factor analysis and Cronbach reliability measurement are the methods used for instrument validation (Straub, 1989; Rosenthal and Rosnow, 1991; Cohen and Swerdlik, 2002).

Through an extensive literature review, only a handful researches with a focus on the psychosocial aspect of PP (Williams, 2000) are found. Consequently, the questionnaire items had to be originally designed for this experiment (Appendix C). For this experiment, the interested constructs are PP satisfaction, PP compatibility, PP communication, and PP confidence. In scoring the answers the pro- (satisfaction, compatibility, communication and confidence) would be weighted from seven ("strongly disagree") to one ("strongly disagree"). For the anti- (satisfaction, compatibility, communication and confidence) questions (4,6,7, 8, 13, 15, 16, 20, 21, 25, 26, 27, 28, 30, 31, 34, 35) the weighting is reversed. Lastly, four short answer items are developed.

The 37 Likert scale question items are checked for validity and reliability by performing factor analysis and Cronbach reliability measurement. The questionnaires from all participated subjects are collected and the results are manually recorded into Microsoft Excel worksheet. The factor analysis was used by means of principal components analysis with varimax rotation. The parameters are 1) eigenvalues greater than one and 2) maximum iteration for convergence was set to be 25. Typically, factor analysis is repeated until all item values are acceptable. For example, after a factor

analysis, any item's value with less than 0.500 is discarded (Straub, 1989) as is any construct with less than minimum three items as well. Only after all item values are 0.500 or higher and all constructs possess a minimum of three questions is the factor analysis complete. Cronbach reliability measurement indicates how well a set of items measures a single latent construct. Typically, 0.700 is viewed as the acceptable minimal value and the higher the value, the more reliable the set is (Cohen and Swerdlik, 2002).

The following Table is the matrix after the first round of factor analysis.

Table 4.8 Rotated Component Matrix I

	Factor Loadings									
Survey Questions	Factor 1	Factor 2	Factor 3	Factor 4	Factor 5	Factor 6	Factor 7	Factor 8	Factor 9	Factor 10
Item 32	.804									
Item 34	.743									
Item 30	.736									
Item 31	.732									
Item 33	.723									
Item 29	.718									
Item 19	.569									
Item 28*										
Item 6		.884								
Item 7		.863								
Item 8		.788								
Item 4		.746								
Item 12		.567								
Item 14		.512								
Item 10			.820							
Item 11			.790							
Item 9			.710							
Item 15*										
Item 13				.767						
Item 26				.575						
Item 20				.570						
Item 16*										
Item 21*										
Item 3					.833					
Item 1					.767					
Item 2					.746					
Item 24						.751				
Item 22						.627				
Item 37							.817			
Item 36							.785			
Item 18								.618		
Item 23								.538		
Item 17								.522		
Item 25*										
Item 27									.803	
Item 35										.742
Item 5										.546

Extraction Method: Principal Component Analysis.
Rotation Method: Varimax with Kaiser Normalization.
a Rotation converged in 24 iterations.
* < 0.500

In validating the constructs, the first exploratory factor analysis is performed on all 37 items and determines which items load together for how many factors (constructs). From the above Table, ten factors are shown. Of those, items that carry a value of less than 0.500 and situations where only two items load together are discarded.

- Items 28, 16, 21, 25 – These items are discarded because they have no value with 0.500 or higher
- Items 24, 22, 37, 36, 27, 35, 5 – These items are discarded because each item doesn't satisfy to the condition of having a minimum of three items that are required to form a set (construct).

This leaves the six factors (factor 1, 2, 3, 4, 5, 8). A second round of factor analysis is performed.

Table 4.9 Rotated Component Matrix II

	Factor Loadings					
Survey Questions	Factor 1	Factor 2	Factor 3	Factor 4	Factor 5	Factor 6
Item 32	.794					
Item 34	.767					
Item 33	.748					
Item 30	.734					
Item 31	.728					
Item 29	.685					
Item 19	.523					
Item 7		.907				
Item 6		.899				
Item 8		.793				
Item 4		.718				
Item 12		.566				
Item 14		.523				
Item 10			.838			
Item 11			.777			
Item 9			.711			
Item 15*						
Item 3				.855		
Item 1				.811		
Item 2				.731		
Item 13					.774	
Item 26					.692	
Item 20					.643	
Item 18						.739
Item 17						.634
Item 23*						

Extraction Method: Principal Component Analysis.
Rotation Method: Varimax with Kaiser Normalization.
a Rotation converged in 8 iterations.
* < .500

From the second round of factor analysis, Items 15, 18, 17, 23 are discarded for the following reasons;

- Items 15, 23 – These items are discarded because they have no value with 0.500 or higher
- Items 18, 17 – These items are discarded because each item doesn't satisfy the condition of having a minimum of three items that are required to form a set (construct).

This leaves five factors (factor 1, 2, 3, 4, 5). A third round of factor analysis is performed.

Table 4.10 Rotated Component Matrix III

	Survey Questions	Factor Loadings				
		PP comm.	PP satisf.	PP conf.	One-Man Prog.	PP compat
32	In PP, my partner's active communication to me allowed me to be more active in expressing my views as well.	.790				
34	In PP, my partner's hand gestures, eye gaze, body positions and other communication cues were NOT used adequately and poorly managed in our communication.	.765				
33	In PP, my partner's voice tone was loud and clear which helped our communication	.742				
30	In PP, my partner did not express nor communicated much (too quiet) which made PP very difficult	.735				
31	In PP, my partner's message delivery was unclear which made PP very difficult	.732				
29	In PP, my partner described his or her point very well and I was able to fully understand	.692				
19	In PP, I really enjoyed my partner's partnership (willing to work as a team, open mind, etc)	.511				
7	I feel that I accomplish more when programming alone		.904			
6	I program best when I'm left alone		.900			
8	Compared to PP, I was able to produce a better solution when I programmed alone		.792			
4	Compared to PP, I enjoyed the one-man programming session more		.719			
12	I enjoyed the PP session more than programming alone		.574			
10	I believe that our code is readable that others can follow and understand it with no problem			.856		
9	I am confident that our work (output) is done correctly.			.791		
11	I believe that our code is efficiently written (with less number of lines).			.783		
14	In PP, I believe that I had an easier time reaching the solutions.			.557		
3	My code is efficiently written (with less number of lines).				.865	
1	I am confident that work (answer) is done correctly.				.790	
2	My code is readable that others can follow and understand with no problem				.761	
13	In PP, My partner and I disagreed frequently in reaching a solution					.785
26	In PP, there were times when I was withdrawn (or maybe upset) because of the disagreements from the partner					.689
20	In PP, my partner insisted in doing things his way and/or did not collaborate					.649

Extraction Method: Principal Component Analysis.
Rotation Method: Varimax with Kaiser Normalization.
a Rotation converged in 6 iterations

In this third round of factor analysis, all items are meeting the requirements. As the matrix Table illustrates, items 32, 34, 33, 30, 31, 29, 19 load to factor 1, items 7, 6, 8, 4, 12 load to factor 2, items 10, 9, 11, 14 load to factor 3, items 3, 1, 2 load to factor 4, and items 13, 26, 20 to factor 5. From the 5 factors or 5 constructs, the appropriate construct names are PP communication for factor 1, PP satisfaction for factor 2, PP confidence for factor 3, programming alone confidence for factor 4, and PP compatibility for factor 5. One-Man Prog. confidence construct is dropped before further analysis. All items conform to their construct name. But item 19 doesn't quite correlate with its construct name, PP communication. Item 19 in the PP communication construct is kept as it's factor analysis and reliability values belong to the construct. The Crobach's Alpha measurement for each set displays above 0.700. One-man prog. confidence construct was advertently not analyzed for any further because only the PP construct items are interested for now. Following is the result;

Table 4.11 Cronbach's Reliability Measurement

Constructs	Reliability (Alpha)
PP comm.	0.8794
PP satisf.	0.8909
PP conf.	0.8314
One-Man Prog. Conf.	0.7833
PP compat.	0.7084

For the inter-judge reliability, a bivariate correlation is performed to assess the inter-judge reliability.

Table 4.12 Inter-Judge Code Productivity Score Correlations

		Judge 1 Code Prod	Judge 2 Code Prod
Judge 1 Code Prod	Pearson Correlation	1.000	.963
	Sig. (2-tailed)	.	.000
	N	125	125
Judge 2 Code Prod	Pearson Correlation	.963	1.000
	Sig. (2-tailed)	.000	.
	N	125	125

Correlation is significant at the 0.01 level (2-tailed).

Table 4.13 Inter-Judge Code Design Score Correlations

		Judge 1 Code Design	Judge 2 Code Design
Judge 1 Code Design	Pearson Correlation	1.000	.967
	Sig. (2-tailed)	.	.000
	N	125	125
Judge 2 Code Design	Pearson Correlation	.967	1.000
	Sig. (2-tailed)	.000	.
	N	125	125

Correlation is significant at the 0.01 level (2-tailed).

As the results show the two judges are highly agreeable with values 0.963 and 0.967 in their scoring in both code productivity and code design.

4.2.3 Hypotheses Evaluated

Before the quantitative analysis begins, determining whether the data qualifies for the parametric tests assumptions or not is imperative (Rosenthal and Rosnow, 1991). There are four criteria that must be checked. They are: 1) Normally distributed data, 2) Homogeneity of variance, 3) Interval data, and 4) Independence (Rowntree, 2003; Wright, 1998; Rosenthal and Rosnow, 1991). The normal distribution refers to the "bell-shaped" distribution of data. Any skewed shaped data will not qualify for the parametric test. The homogeneity of variance means that the variances should not change systematically throughout the data. Interval data means data should be measured at least at the interval level. Independence means that data from different subjects are independent, the behavior of one participant does not influence the behavior of another. Both interval data and independence are analyzed by common sense or visual examination.

For the normally distributed data and homogeneity of variance, there are more scientific ways for analysis. As is the practice with many researchers, the most common method of checking normality is, and unfortunately still is, by visual examination of either a histogram or a Q-Q graph. This subjective practice may discredit the experimental findings and also present challenges. Striving for more accurate analysis, two normality tests, Kolmogorov-Smirnov and Shapiro-Wilk tests can be used. These tests compare the set of scores in the sample to a normally distributed set of scores with the same mean and standard deviation. If the test results are non-significant ($p > 0.05$), then the sample data is normally distributed. The following is the normality test result of code productivity and code design of [pairtype].

Table 4.14 Normal Distribution of Dependent Variables

			Statistic	Std. Error
Code Productivity	Mean		5.4512	.2684
	95% Confidence Interval for Mean	Lower Bound	4.9198	
		Upper Bound	5.9826	
	5% Trimmed Mean		5.5111	
	Median		5.5000	
	Variance		8.862	
	Std. Deviation		2.9770	
	Minimum		.00	
	Maximum		10.00	
	Range		10.00	
	Interquartile Range		4.5000	
	Skewness		**-.320**	.218
	Kurtosis		-1.045	.433
Code Design	Mean		5.4289	.2638
	95% Confidence Interval for Mean	Lower Bound	4.9067	
		Upper Bound	5.9511	
	5% Trimmed Mean		5.4862	
	Median		6.0000	
	Variance		8.559	
	Std. Deviation		2.9256	
	Minimum		.00	
	Maximum		10.00	
	Range		10.00	
	Interquartile Range		5.0000	
	Skewness		**-.388**	.218
	Kurtosis		-.934	.433

The negative Skewness values of both code productivity (-0.320) and code design (-0.388) indicate an overabundance of scores on the right side of the distribution. For a normal distribution, the Skewness value should be zero.

Table 4.15 Tests of Normality of Dependent Variables

	Kolmogorov-Smirnov (K-S) Test		
Dependent variable	Statistic	df	Sig.
Code Productivity	.145	123	.000
Code Design	.118	123	.000

a Lilliefors Significance Correction

Here, K-S test shows code productivity is significant ($p < 0.001$) and code design is also significant ($p < 0.001$). Therefore code productivity and code design are not normally distributed. But from the histogram and normal Q-Q plots of both dependent variables (Appendix N), code productivity and code design do not exhibit any significant deviations from normality. However, the K-S test result shows otherwise.

The K-S test result ($p < 0.001$) clearly indicates that the data is not fit for parametric tests. However, there is an option of data transformation. In an effort to transform the data, the authors have tried following list of transformation options:

1) Logarithmic transformation

Y = log10 (x)
Y = log10 (x+1)
Y = log (x)
Y = log (x+1)

2) Square Root transformation

Y = square root (x)

3) Reciprocal transformation

Y = 1 / X
Y = 1 / (X+1)

4) Exponential transformation

Y = Exponential (x)
Y = 2 ^ X

5) Power transformation

Y = X ^ 2
Y = X ^ 3

6) Arcsine transformation

Y = Arcsine (X)
Y = Arcsine (X/100)

However, the results were not normally distributed. This failure of normality on for both dependent variables directs to further analysis using non-parametric tests. The most appropriate non-parametric test for this situation appears to be the Kruskal-Wallis H (K-W) test. The K-W test, an extension of the Mann-Whitney U test, is the nonparametric analog of a one-way analysis of variance (ANOVA) and detects differences in distribution location. It also assumes that there is no a priori ordering of the k populations from which the samples are drawn. For the statistics computer software analysis tool, SPSS statistics software application version 9.0 for windows (www.spss.com) is used.

[Pairtype]

H 1: The pairs who are alike in their kind of perception or judgment (dominant and auxiliary types only) but not both, [divrs], would achieve significantly higher **[score]** than the pairs who are alike in both, [alike]:

H 1.1: **[score]** – code productivity

H 1.2: **[score]** – code design

H 2: The pairs who are alike in their kind of perception or judgment (dominant and auxiliary types only) but not both, [divrs], would achieve significantly higher **[score]** than the pairs who are opposite in both [opp]:

H 2.1: **[score]** – code productivity

H 2.2: **[score]** – code design

Table 4.16 Descriptive of [pairtype]

Dependent variable	[pairtype]	N	Mean	Std. Deviation
Code productivity	[opp]	40	5.31	3.19
	[divrs]	40	6.46	2.43
	[alike]	43	4.64	3.03
Code design	[opp]	40	5.24	3.05
	[divrs]	40	6.49	2.35
	[alike]	43	4.62	3.05

From Table 4.16, [divrs] shows the highest mean values (6.46, 6.49) with the lowest standard deviation values (2.43, 2.35).

Table 4.17 Test Statistics of [pairtype]

	Code productivity	Code design
Chi-Square	7.067	7.710
df	2	2
Asymp. Sig.	.029	.021

Kruskal Wallis Test
Grouping Variable: [pairtype]

From Table 4.17, both code productivity ($p < 0.05$) and code design ($p < 0.05$) show significance. However, this does not show which groups are significantly different from which of the others. To find out, Mann-Whitney test is performed between [divrs] vs. [opp], [divrs] vs. [alike], and [opp] vs. [alike].

Table 4.18 Test Statistics of [divrs] vs. [opp]

	Code productivity	Code design
Mann-Whitney U	650.5	629.5
Wilcoxon W	1470.5	1449.5
Z	-1.443	-1.645
Asymp. Sig. (2-tailed)	.149	.100
Asymp. Sig. (1-tailed)	.075	.050

Grouping Variable: [pairtype]

From Table 4.18, two-tailed significance value is shown. However, this needs to be changed to one-tailed significance value. The two-tailed significance value is can be used as it is when no prediction has been made about which group will differ from which (Wright, 1998, Rowntree, 2003). However, hypothesis 1 states that [divrs] would achieve significantly higher mean than [alike], therefore using one-tailed probability is appropriate (Field, 2003). This value can be obtained by taking the two-tailed value and dividing it by two. By 5% significance level, [divrs] is not significantly higher than [opp] in code productivity mean. But [divrs] is significantly higher than [opp] in code design mean. By 10% significance level, [divrs] is significantly higher than [opp] in both code productivity and code design means.

Table 4.19 Test Statistics of [divrs] vs. [alike]

	code productivity	code design
Mann-Whitney U	560.5	553.0
Wilcoxon W	1506.5	1499.0
Z	-2.737	-2.806
Asymp. Sig. (2-tailed)	.006	.005
Asymp. Sig. (1-tailed)	.003	.003

Grouping Variable: [pairtype]

From Table 4.19, [divrs] is significantly higher than [alike] in both code productivity and code design means. . Not included in the hypotheses, yet [opp] vs. [alike] is performed below.

Table 4.20 Test Statistics of [opp] vs. [alike]

	code productivity	code design
Mann-Whitney U	752.5	755.0
Wilcoxon W	1698.5	1701.0
Z	-.983	-.960
Asymp. Sig. (2-tailed)	.326	.337
Asymp. Sig. (1-tailed)	.163	.169

Grouping Variable: [pairtype]

From Table 4.20, both code productivity and code design significance value indicate that [opp] is not significantly differ from [alike].

H 1.1: **[score]** – code productivity ***Supported***

H 1.2: **[score]** – code design ***Supported***

H 2.1: **[score]** – code productivity ***Supported***

H 2.2: **[score]** – code design ***Supported***

[Paircomm]

H 3: The high communication skill level pairs, [HH], would yield a significantly higher **[score]** than the mixed communication skill level pairs, [HL].

H 3.1: **[score]** – code productivity

H 3.2: **[score]** – code design

H 4: The high communication skill level pairs, [HH], would yield a significantly higher **[score]** than the low communication skill level pairs, [LL].

H 4.1: **[score]** – code productivity

H 4.2: **[score]** – code design

Table 4.21 Descriptive of [paircomm]

Dependent variable	[paircomm]	N	Mean	Std. Deviation
code productivity	[HH]	14	4.79	3.33
	[HL]	45	5.78	2.83
	[LL]	15	5.97	2.72
code design	[HH]	14	5.07	3.01
	[HL]	45	5.64	2.74
	[LL]	15	5.75	2.94

Table 4.22 Test Statistics of [paircomm]

	code productivity	code design
Chi-Square	1.330	.369
df	2	2
Asymp. Sig.	.514	.832

Kruskal Wallis Test
Grouping Variable: [paircomm]

From the K-W test, the result shows that there are no significant differences between the means of the three groups in both code productivity ($p = 0.514$) and code design ($p = 0.832$). Therefore, further group contrasts are insignificant.

H 3.1: **[score]** – code productivity ***Not Supported***

H 3.2: **[score]** – code design ***Not Supported***

H 4.1: **[score]** – code productivity ***Not Supported***

H 4.2: **[score]** – code design ***Not Supported***

[Pairgender]

H 5: Male-Female pairs, [MF], would significantly achieve a higher **[score]** than Male-Male pairs, [MM].

H 5.1: **[score]** – code productivity

H 5.2: **[score]** – code design

H 6: Male-Female pairs, [MF], would significantly achieve a higher **[score]** than Female-Female pairs, [FF].

H 6.1: **[score]** – code productivity

H 6.2: **[score]** – code design

Table 4.23 Descriptive of [pairgender]

Dependent variable	[pairgender]	N	Mean	Std. Deviation
code productivity	[MM]	67	5.60	2.95
	[MF]	44	5.44	3.03
	[FF]	12	4.63	3.04
code design	[MM]	67	5.60	2.88
	[MF]	44	5.42	3.01
	[FF]	12	4.50	2.92

Table 4.24 Test Statistics of [pairgender]

	code productivity	code design
Chi-Square	1.093	1.630
df	2	2
Asymp. Sig.	.579	.443

Kruskal Wallis Test
Grouping Variable: [pairgender]

Again from the K-W test, the result shows that there are no significant differences between the means of the three groups in both code productivity ($p = 0.579$) and code design ($p = 0.443$). Further group contrasts are not performed.

H 5.1: **[score]** – code productivity ***Not Supported***

H 5.2: **[score]** – code design ***Not Supported***

H 6.1: **[score]** – code productivity ***Not Supported***

H 6.2: **[score]** – code design ***Not Supported***

Similar to the normal data distribution check on the quantitative result, the questionnaire result is also checked for its normal data distribution.

Table 4.25 Descriptive of Constructs

		PP Comm.	PP Satisf.	PP Conf.	PP Compat.
N	Valid	218	217	220	223
	Missing	5	6	3	0
Mean		3.67E-17	8.19E-18	1.21E-16	2.19E-17
Median		.121	.185	.115	.292
Mode		1.68	.504	.944	1.20
Std. Deviation		1.00	1.00	1.00	1.00
Variance		1.00	1.00	1.00	1.00
Skewness		-.966	-.298	-.646	-.879
Range		5.52	4.22	4.86	4.26
Minimum		-3.84	-2.31	-3.10	-3.06
Maximum		1.68	1.91	1.75	1.20

The negative Skewness values (-0.966, -0.298, -0.646, and -0.879) indicate an overabundance of scores on the right side of the distribution. Again, for a normal distribution, the Skewness value should be zero.

Table 4.26 Tests of Normality of Constructs

	Kolmogorov-Smirnov (K-S) test		
Construct	Statistic	df	Sig.
PP comm.	.112	210	.000
PP Satisf.	.082	210	.002
PP Conf.	.059	210	.076
PP Compat.	.121	210	.000

Based on the normality test, Table 4.26, only PP confidence is showing normal distribution ($p = 0.076$). However, all constructs are to analyze by non-parametric test for a consistency.

H 7: The pairs who are alike in their kind of perception or their kind of judgment (dominant and auxiliary types) but not both, [divrs], would achieve a significantly higher level than the pairs who are alike in both, [alike] from following constructs:

H 7.1: PP communication

H 7.2: PP satisfaction

H 7.3: PP confidence

H 7.4: PP compatibility

H 8: The pairs who are alike in their kind of perception or their kind of judgment (dominant and auxiliary types) but not both, [divrs], would achieve a significantly higher level than the pairs who are opposite in both, [opp] from following constructs:

H 8.1: PP communication

H 8.2: PP satisfaction

H 8.3: PP confidence

H 8.4: PP compatibility

Table 4.27 Test Statistics of [pairtype] Constructs

	PP Comm.	PP Satisf.	PP Conf.	PP Compat.
Chi-Square	2.33	4.42	3.99	2.37
Df	2	2	2	2
Asymp. Sig.	.312	.109	.136	.305

Kruskal Wallis Test
Grouping Variable: [pairtype]

From the K-W test, the result shows that there are no significant differences between the means of the three groups in PP communication ($p = 0.312$), PP satisfaction ($p = 0.109$), PP confidence ($p = 0.136$), and PP compatibility ($p = 305$). Therefore group contrasts are not performed.

H 7.1: PP communication ***Not Supported***

H 7.2: PP satisfaction ***Not Supported***

H 7.3: PP confidence ***Not Supported***

H 7.4: PP compatibility ***Not Supported***

H 8.1: PP communication ***Not Supported***

H 8.2: PP satisfaction ***Not Supported***

H 8.3: PP confidence ***Not Supported***

H 8.4: PP compatibility ***Not Supported***

H 9: The high communication skill pairs, [HH], would yield a significantly higher level than the mixed communication skill pairs, [HL] from following constructs:

H 9.1: PP communication

H 9.2: PP satisfaction

H 9.3: PP confidence

H 9.4: PP compatibility

H 10: The high communication skill pairs, [HH], would yield a significantly higher level than the low communication skill pairs, [LL] from following constructs;

H 10.1: PP communication

H 10.2: PP satisfaction

H 10.3: PP confidence

H 10.4: PP compatibility

Table 4.28 Test Statistics of [paircomm] Constructs

	PP Comm.	PP Satisf.	PP Conf.	PP Compat.
Chi-Square	6.436	3.053	.798	.884
Df	2	2	2	2
Asymp. Sig.	.040	.217	.671	.643

Kruskal Wallis Test
Grouping Variable: [paircomm]

From the K-W test, Table 4.28, with regards to PP satisfaction ($p = 0.217$), PP confidence ($p = 0.671$), and PP compatibility ($p = 0.643$) the means of the three groups are not significantly different from each other, while with PP communication does they are ($p < 0.05$). In order to determine which ones are significantly different from which others, a Mann-Whitney test is performed on [paircomm] groups.

Table 4.29 Test Statistics [HH] vs. [HL]: [paircomm]-PP comm.

	PP comm.
Mann-Whitney U	763.0
Wilcoxon W	4166.0
Z	-1.93
Asymp. Sig. (2-tailed)	.054
Asymp. Sig. (1-tailed)	.027

a Grouping Variable: [paircomm]

From Table 4.29, [HH] exhibits significantly higher mean ($p(1\text{-tailed}) < 0.05$) than [HL] from PP comm. construct.

Table 4.30 Test Statistics [HH] vs. [LL]: [paircomm]-PP comm.

	PP comm.
Mann-Whitney U	179.000
Wilcoxon W	479.000
Z	-2.420
Asymp. Sig. (2-tailed)	.016
Asymp. Sig. (1-tailed)	.008

a Grouping Variable: [paircomm]

From Table 4.30, again [HH] exhibits significantly higher mean ($p(1\text{-tailed}) < 0.01$) than [LL] from PP comm. construct.

Table 4.31 Test Statistics [HL] vs. [LL]: [paircomm]-PP comm.

	PP comm.
Mann-Whitney U	824.5
Wilcoxon W	1124.5
Z	-1.204
Asymp. Sig. (2-tailed)	.229
Asymp. Sig. (1-tailed)	.115

a Grouping Variable: [paircomm]

From Table 4.31, [HL] do not exhibit significantly difference (p = 0.115) from [LL] in mean.

H 9.1: PP communication	***Supported***
H 9.2: PP satisfaction	***Not Supported***
H 9.3: PP confidence	***Not Supported***
H 9.4: PP compatibility	***Not Supported***

H 10.1: PP communication	***Supported***
H 10.2: PP satisfaction	***Not Supported***
H 10.3: PP confidence	***Not Supported***
H 10.4: PP compatibility	***Not Supported***

H 11: Male-Female pairs, [MF], would significantly achieve a higher level than Male-Male pairs, [MM] from following constructs;

H 11.1: PP communication

H 11.2: PP satisfaction

H 11.3: PP confidence

H 11.4: PP compatibility

H 12: Male-Female pairs, [MF], would significantly achieve a higher level than Female-Female pairs, [FF] from following constructs;

H 12.1: PP communication

H 12.2: PP satisfaction

H 12.3: PP confidence

H 12.4: PP compatibility

Table 4.32 Test Statistics of [pairgender] Constructs

	PP Comm.	PP Satisf.	PP Conf.	PP Compat.
Chi-Square	12.105	5.745	.984	19.209
df	2	2	2	2
Asymp. Sig.	.002	.057	.612	.000

Kruskal Wallis Test
Grouping Variable: [pairgender]

From Table 4.32, K-W test, PP communication ($p < 0.05$), PP satisfaction ($p < 0.1$), and PP compatibility ($p < 0.001$) show significant differences among the three groups of [pairgender]. For each construct, following group contrasts are performed.

Table 4.33 Test Statistics [MM] vs. [MF]:[pairgender]-PP comm.

	PP comm.
Mann-Whitney U	3596.5
Wilcoxon W	6836.5
Z	-2.921
Asymp. Sig. (2-tailed)	.003
Asymp. Sig. (1-tailed)	.002

a Grouping Variable: [pairgender]

From Table 4.33, [MM] shows significantly higher mean ($p < 0.01$) than [MF].

Table 4.34 Test Statistics [MF] vs. [FF]:[pairgender]-PP comm.

	PP comm.
Mann-Whitney U	444.0
Wilcoxon W	3684.0
Z	-2.808
Asymp. Sig. (2-tailed)	.005
Asymp. Sig. (1-tailed)	.003

Grouping Variable: [pairgender]

From Table 4.34, [FF] shows significantly higher mean ($p < 0.01$) than [MF].

Table 4.35 Test Statistics [MM] vs. [FF]:[pairgender]-PP comm.

	PP comm.
Mann-Whitney U	958.5
Wilcoxon W	8098.5
Z	-1.063
Asymp. Sig. (2-tailed)	.288
Asymp. Sig. (1-tailed)	.144

a Grouping Variable: [pairgender]

From Table 4.35, there is no significance in means ($p = 0.144$) between [MM] and [FF].

Table 4.36 Test Statistics [MM] vs. [MF]:[pairgender]-PP satisf.

	PP satisf.
Mann-Whitney U	3819.5
Wilcoxon W	6900.5
Z	-2.184
Asymp. Sig. (2-tailed)	.029
Asymp. Sig. (1-tailed)	.015

a Grouping Variable: [pairgender]

From Table 4.36, [MM] exhibits significantly higher mean ($p < 0.05$) than [MF].

Table 4.37 Test Statistics [MF] vs. [FF]:[pairgender]-PP satisf.

	PP satisf.
Mann-Whitney U	564.5
Wilcoxon W	3645.5
Z	-1.604
Asymp. Sig. (2-tailed)	.109
Asymp. Sig. (1-tailed)	.054

a Grouping Variable: [pairgender]

From Table 4.37, [FF] shows significantly higher mean ($p < 0.1$) than [MF].

Table 4.38 Test Statistics [MM] vs. [FF]:[pairgender]-PP satisf.

	PP satisf.
Mann-Whitney U	1039.0
Wilcoxon W	8299.0
Z	-.619
Asymp. Sig. (2-tailed)	.536
Asymp. Sig. (1-tailed)	.268

a Grouping Variable: [pairgender]

From Table 4.38, [FF] do not show significant difference from [MM].

Table 4.39 Test Statistics [MM] vs. [MF]:[pairgender]-PP compat.

	PP compat.
Mann-Whitney U	3255.0
Wilcoxon W	6576.0
Z	-4.123
Asymp. Sig. (2-tailed)	.000
Asymp. Sig. (1-tailed)	.000

a Grouping Variable: [pairgender]

From Table 4.39, [MM] exhibits significantly higher mean ($p < 0.001$) than [MF].

Table 4.40 Test Statistics [MF] vs. [FF]:[pairgender]-PP compat.

	PP compat.
Mann-Whitney U	482.000
Wilcoxon W	3803.000
Z	-2.797
Asymp. Sig. (2-tailed)	.005
Asymp. Sig. (1-tailed)	.003

a Grouping Variable: [pairgender]

From Table 4.40, [FF] shows significantly higher mean ($p < 0.05$) than [MF].

Table 4.41 Test Statistics [MM] vs. [FF]:[pairgender]-PP compat.

	PP compat.
Mann-Whitney U	1136.000
Wilcoxon W	8639.000
Z	-.495
Asymp. Sig. (2-tailed)	.621
Asymp. Sig. (1-tailed)	.210

a Grouping Variable: [pairgender]

From Table 4.41, [FF] and [MM] do not show significant difference (p = 0.210) to each other.

H 11.1: PP communication	***Not Supported***
H 11.2: PP satisfaction	***Not Supported***
H 11.3: PP confidence	***Not Supported***
H 11.4: PP compatibility	***Not Supported***
H 12.1: PP communication	***Not Supported***
H 12.2: PP satisfaction	***Not Supported***
H 12.3: PP confidence	***Not Supported***
H 12.4: PP compatibility	***Not Supported***

4.2.4 One-man Programming vs. Pair Programming

The analysis of one-man programming versus pair programming is set forth in the following.

Table 4.42 Tests of Normality One-man Programming

Dependent variable	Kolmogorov-Smirnov (K-S) Test		
	Statistic	df	Sig.
Code Productivity	.063	121	.200
Code Design	.060	121	.200

Lilliefors Significance Correction

Here, K-S test shows code productivity is insignificant ($p > 0.05$) and code design is also insignificant ($p > 0.05$). Therefore code productivity and code design are normally distributed. Parametric test (t-test) is used for further analysis.

Table 4.43 Descriptives Statistics All [pairtype] Combined

Measurement	Mode	N	Mean	Std. Deviation
Code Productivity	One-man prog.	41	5.46	1.60
	Pair prog.	41	5.85	2.08
Code Design	One-man prog.	41	4.94	1.46
	Pair prog.	41	5.91	1.95

In comparing the mean values of one-man programming and PP, the PP mean value shows higher value in both code productivity (+0.31) and code design (+0.97).

Table 4.44 Paired Samples Correlations All [pairtype] Combined

Measurement	Comparison	N	Correlation	Sig.
Code Productivity	One-man prog. vs. Pair prog.	41	.409	.008
Code Design	One-man prog. vs. Pair prog.	41	.276	.080

In the code productivity comparison, a fairly large correlation coefficient ($r = 0.409$) indicates the one-man programming and PP are significantly correlated ($p < 0.05$) in code productivity. However, in code design, no sign of correlation is shown ($r = 0.276$, $p > 0.05$).

Table 4.45 Paired Samples Test All [pairtype] Combined

		Paired Differences							
					95% Confidence Interval of the Difference				
		Mean	Std. Deviation	Std. Error Mean	Lower	Upper	T	df	Sig. (2-tailed)
Code Prod.	One-man prog. vs. Pair prog.	-.39	2.04	.32	-1.03	.254	-1.23	40	.228
Code Design	One-man Prog. vs. Pair prog.	-.98	2.09	.33	-1.64	-.316	-2.99	40	.005

The t-value of one-man programming versus PP code productivity comparison (-1.23) indicates that one-man programming had a smaller mean than PP, and the difference is not significant ($p > 0.05$). However, the t-value code design comparison (-2.99) shows that the difference between the mean values is significant ($p < 0.05$). In other words, the qualitative measure between one-man programming and PP shows that the PP team exhibits a significant higher score, or performed significantly better in code design, code efficiency and code readability than one-man programming.

Opposite Type [opp]

The analysis of one-man programming versus pair programming of [opp] type is set forth in the following.

Table 4.46 Paired Samples Statistics [opp] Type

Measurement	Mode	N	Mean	Std. Deviation
Code Productivity	One-man Programming	15	5.90	1.68
	Pair prog.	15	5.42	2.51
Code Design	One-man Design	15	5.11	1.40
	Pair prog.	15	5.33	2.37

For the opposite type matched pairs, the comparison of mean values of one-man programming and PP shows mixed results. In the code productivity, one-man programming showed a higher mean value (+0.48) than PP, but in the code design, PP showed a higher mean value (+0.22) than one-man programming.

Table 4.47 Paired Samples Correlations [opp] Type

Measurement	Mode	N	Correlation	Sig.
Code Productivity	One-man programming vs. Pair prog.	15	.663	.007
Code Design	One-man programming vs. Pair prog.	15	.279	.313

In the code productivity comparison, a fairly large correlation coefficient (r = 0.663) indicates the one-man programming and PP are significantly correlated ($p < 0.05$) in code productivity. However, in code design, no sign of correlation is shown (r = 0.279, $p > 0.05$).

Table 4.48 Paired Samples Test [opp] Type

		Paired Differences					t	df	Sig. (2-tailed)
		Mean	Std. Deviation	Std. Error Mean	95% Confidence Interval of the Difference Lower	Upper			
Code Prod.	One-man programming vs. Pair programming	.48	1.88	.49	-.56	1.52	.985	14	.341
Code Design	One-man programming vs. Pair programming	-.22	2.39	.62	-1.54	1.11	-.350	14	.732

The t value of code productivity comparison (t = 0.985) indicates that one-man programming's higher mean value than PP, but not significant ($p > 0.05$). The t value of code design comparison (-0.350) indicates that PP shows a higher mean value than one-man programming, and again, not significant ($p > 0.05$).

Diverse Type [divrs]

The analysis of one-man programming versus pair programming of [divrs] type is set forth in the following.

Table 4.49 Paired Samples Statistics [divrs] Type

Measurement	Mode	N	Mean	Std. Deviation
Code Productivity	One-man Prog.	14	5.05	1.60
	Pair Prog.	14	6.82	2.01
Code Design	One-man Prog.	14	4.62	1.45
	Pair Prog.	14	6.84	1.68

For the diverse type matched pairs, the mean value of PP is convincingly higher than one-man programming in both code productivity (+1.77) and code design (+2.22).

Table 4.50 Paired Samples Correlations [divrs] Type

Measurement	Mode	N	Correlation	Sig.
Code Productivity	One-man programming vs. Pair programming	14	.402	.154
Code Design	One-man programming vs. Pair programming	14	.430	.125

The large correlation coefficients of both code productivity (r = 0.402) and code design (r = 0.430) indicate no sign of correlation between one-man programming (p > 0.05) and PP (p > 0.05).

Table 4.51 Paired Samples Test [divrs] Type

		Paired Differences					t	df	Sig. (2-tailed)
		Mean	Std. Dev	Std. Error Mean	95% Confidence Interval of the Difference Lower	Upper			
Code Prod.	One-man prog vs. Pair prog.	-1.77	2.00	.54	-2.93	-.62	-3.31	13	.006
Code Design	One-man prog. vs. Pair prog	-2.22	1.68	.45	-3.19	-1.25	-4.94	13	.000

The t values of both code productivity and code design indicate that PP shows higher mean values than one-man programming. Furthermore, these mean values of PP are significantly higher than one-man programming in both code productivity (p < 0.05) and code design (p < 0.05).

Alike Type [alike]

The analysis of one-man programming versus pair programming of [alike] type is set forth in the following.

Table 4.52 Paired Samples Statistics [alike] Type

Measurement	Mode	N	Mean	Std. Deviation
Code Productivity	One-man Prog.	12	5.48	1.50
	Pair Prog.	12	5.69	1.26
Code Design	One-man Prog.	12	5.02	1.63
	Pair Prog.	12	5.88	1.33

For the alike type matched pairs, the mean value of PP is higher than one-man programming in both code productivity (+0.21) and code design (+0.86).

Table 4.53 Paired Samples Correlations [alike] Type

		N	Correlation	Sig.
Code Productivity	One-man programming vs. Pair programming	12	.416	.178
Code Design	One-man programming vs. Pair programming	12	.440	.152

The large correlation coefficients of both code productivity (r = 0.416) and code design (r = 0.440) indicate no sign of correlation between one-man programming (p > 0.05) and PP (p > 0.05).

Table 4.54 Paired Samples Test [alike] Type

		Paired Differences					t	df	Sig. (2-tailed)
		Mean	Std. Deviation	Std. Error Mean	95% Confidence Interval of the Difference Lower	Upper			
Code Prod.	One-man prog vs. Pair prog.	-.21	1.50	.43	-1.16	.75	-.475	11	.644
Code Design	One-man prog. vs. Pair prog	-.85	1.59	.46	-1.86	.16	-1.856	11	.090

The t values of both code productivity (t = -0.475) and code design (t = -1.856) indicate that PP shows higher mean values than one-man programming. However, these differences are not significant in both code productivity (p > 0.05) and code design (p > 0.05).

4.2.5 Interaction Effects

The interaction effects between the independent variables were also checked for any saliencies. Following is the [pairtype] vs [pairgender] interaction analysis.

Table 4.55 Descriptive Statistics of [pairtype] and [pairgender]

[pairgender]	[pairtype]	Mean	Std. Deviation	N
[MM]	[opp]	5.52	3.36	21
	[divrs]	6.65	2.26	23
	[alike]	4.63	2.94	23
[MF]	[opp]	4.73	3.23	13
	[divrs]	6.65	2.62	13
	[alike]	5.08	3.06	18
[FF]	[opp]	5.83	2.80	6
	[divrs]	4.75	2.72	4
	[alike]	.75	1.06	2

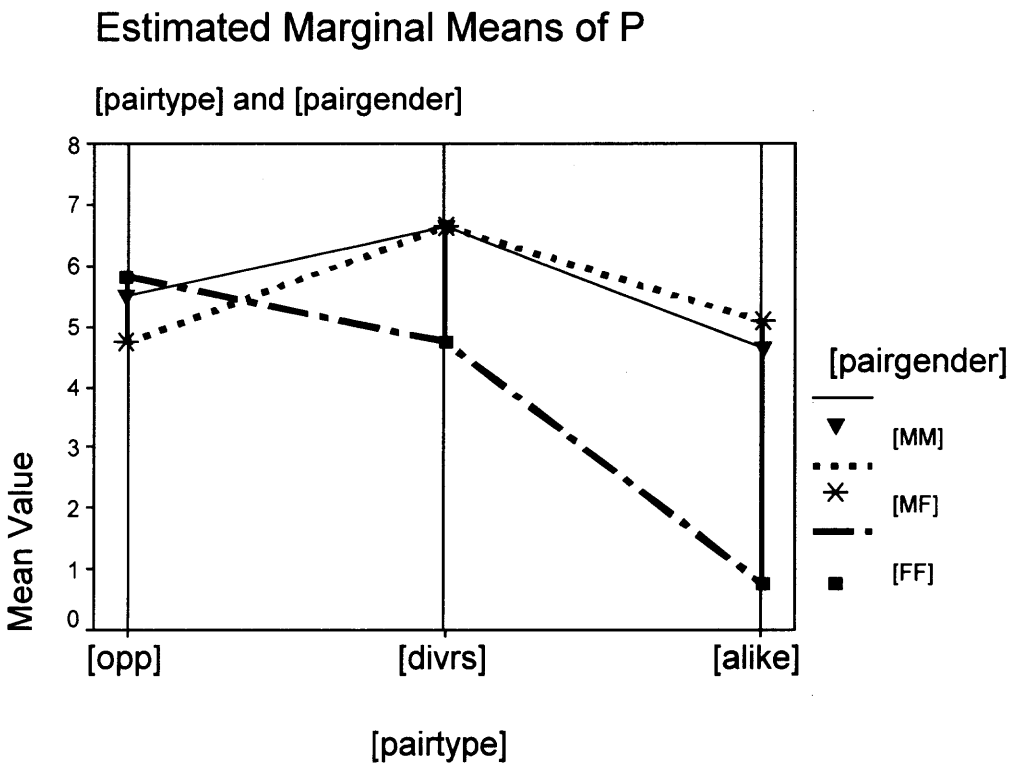


Figure 4.2 [pairtype] and [pairgender]

From Figure 4.2, the pairs that are [divrs] and either [MM] or [MF] show the highest mean and pairs that are [alike] and [FF] show the lowest mean. However, the low N of [FF] cautions the finding. One interesting observation is that [FF] tops [MM] and [MF] when they are [opp], but they drastically drop in their performance when they are [divrs] and [alike]. Also, only the [FF] curve is showing a decline from left to right, [opp] to [alike]. In summary, one would achieve the highest productivity from either [divrs] and [MM] or [divrs] and [MF]. Also, overall, [FF] tends to produce a lower output than the other two.

Table 4.56 Descriptive Statistics Dependent Variable: Code Design

[pairgender]	[pairtype]	Mean	Std. Deviation	N
[MM]	[opp]	5.58	3.21	21
	[divrs]	6.61	2.26	23
	[alike]	4.61	2.90	23
[MF]	[opp]	4.62	3.19	13
	[divrs]	6.62	2.58	13
	[alike]	5.14	3.06	18
[FF]	[opp]	5.42	2.35	6
	[divrs]	5.38	2.39	4
	[alike]	.00	.00	2

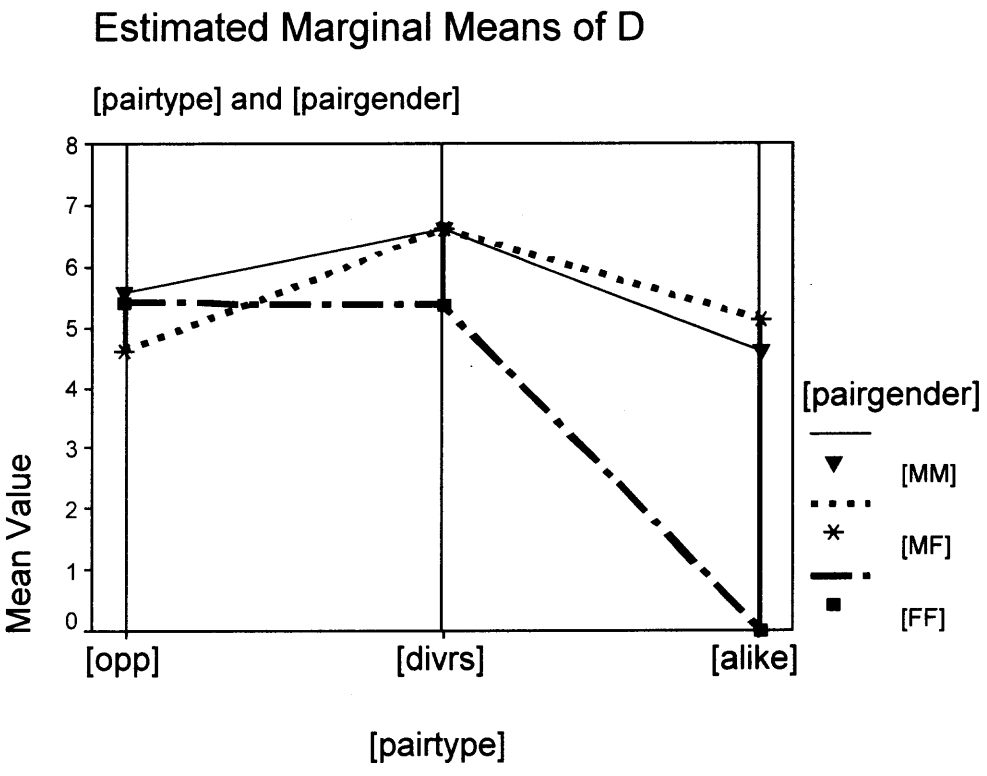


Figure 4.3 [pairtype] and [pairgender] by code design

Again, a very similar representation to the productivity dependent variable occurs, as the pairs with [divrs] and either [MM] or [MF] show the highest mean, while the pairs with [alike] and [FF] show the lowest mean. The difference between the pairs with [divrs] and [MM] and also [divrs] and [MF] to the pairs with [divrs] and [FF] in code design is shorter than the same pairs in code productivity. The [opp] and [FF] pairs are almost equal in mean to the [opp] and [MM] pairs. Unlike with the code productivity, the [FF] curve does not quite show the declination here. For the interaction effects of [pairgender] vs. [paircomm] and [pairtype] vs. [paircomm], not enough samples are available for the analysis. For example, there are no samples that fall under [divrs] and also [HH]. Therefore these interaction analyses are not available.

4.2.6 Hypotheses Summary

Table 4.57 Hypotheses 1-6 Summary

	Main Hypotheses			Sub-Hypotheses	Supported	Not Supported
MBTI Personality	**1**	[divrs] would achieve significantly higher **[score]** than [alike]	1.1	code productivity	√	
			1.2	code design	√	
	2	[divrs] would achieve significantly higher **[score]** than [opp]	2.1	code productivity	√	
			2.2	code design	√	
Pair Communication	**3**	[HH] would yield a significantly higher **[score]** than [HL]	3.1	code productivity		√
			3.2	code design		√
	4	[HH] would yield a significantly higher **[score]** than [LL].	4.1	code productivity		√
			4.2	code design		√
Pair Gender	**5**	[MF] would significantly achieve a higher **[score]** than [MM]	5.1	code productivity		√
			5.2	code design		√
	6	[MF] would significantly achieve a higher **[score]** than [FF]	6.1	code productivity		√
			6.2	code design		√

Table 4.58 Hypotheses 7-12 Summary

	Main Hypotheses			Sub-Hypotheses	Supported	Not Supported
MBTI Personality	**7**	[divrs] would achieve a significantly higher level than [alike]	7.1	PP communication		√
			7.2	PP satisfaction		√
			7.3	PP confidence		√
			7.4	PP compatibility		√
	8	[divrs] would achieve a significantly higher level than [opp]	8.1	PP communication		√
			8.2	PP satisfaction		√
			8.3	PP confidence		√
			8.4	PP compatibility		√
Pair Communication	**9**	[HH] would yield a significantly higher level than [HL]	9.1	PP communication	√	
			9.2	PP satisfaction		√
			9.3	PP confidence		√
			9.4	PP compatibility		√
	10	[HH] would yield a significantly higher level than [LL]	10.1	PP communication	√	
			10.2	PP satisfaction		√
			10.3	PP confidence		√
			10.4	PP compatibility		√
Pair Gender	**11**	[MF] would achieve a significantly higher level than [MM]	11.1	PP communication		√
			11.2	PP satisfaction		√
			11.3	PP confidence		√
			11.4	PP compatibility		√
	12	[MF] would significantly achieve a higher level than [FF]	12.1	PP communication		√
			12.2	PP satisfaction		√
			12.3	PP confidence		√
			12.4	PP compatibility		√

4.2.7 Discussion

Based on the results for hypotheses 1 and 2, the theory on "diversity for better design and higher productivity" appears to hold up well in the PP context. The clear differences between the groups' output show that the MBTI type indicator is a significant factor that one may manipulate and optimize in cognitive-intensive tasks. Interestingly, in the comparison between [opp] vs. [alike], the mean difference is not significant and this fact further substantiates the "diversity for better design and productivity" theory. However, in the PP experience measurement, there are no clear differences among the groups. This finding may lead to an assumption that there is a separation between cognitive interaction and one's emotional experience. Or in other words, a propitious cognitive interaction on a job between two individuals or groups does not always lead to or reach a high level of satisfaction, compatibility, communication, and confidence.

With regards to communication skill level measurement, design and productivity were not impacted. As expected, the [HH] pairs exhibited significantly different results and a higher level of communication than [HL] and [LL], but did not show differences with regards to satisfaction, confidence and compatibility. This result contradicts both today's common belief and the field survey result which establish that clear and well-delivered message exchanges and communication between two individuals do not always assure better design, higher productivity, a high level of satisfaction, confidence and compatibility. Conversely, a pair having a high level of communication between them does not necessarily experience a high level of satisfaction, compatibility or confidence on a job completed.

Similar to the survey result, gender appears not to be a factor in design and productivity. However, contrary to the result of [pairtype], the level of experience does

show significant differences in communication, satisfaction and compatibility. Both [MM] and [FF] showed higher levels of communication, satisfaction and compatibility than [MF]. This is an interesting finding on gender factor. Most professionals and even some researchers, believe that gender factor has absolutely no impact nor influences in situations such as PP. But this experiment's finding shows a different version. Based on this experiment, the gender factor exhibited no impact on design and productivity, but its highly significant differences in both communication, satisfaction and compatibility illustrate a substantial affinity between same gender pairs. [MM] and [FF] enjoy each other as a partner in situations such as PP much more than [MF], but they both do not necessarily produce better design and higher productivity nor have a high level of confidence of their finished product.

As a part of the questionnaire, comments were also collected from the subjects. A number of profound comments were noticed from female subjects. The following are some comments from the female subjects of [FF]:

> *"My partner was of the same sex (we are both female). I think this really helped because it made us feel more comfortable around each other. I really think that if I had worked with a guy, there would have been some initial awkwardness. Whether this would have been counter-productive or not would depend on the person and maybe if they came from a background that doesn't respect a woman's opinion".*

> *"I think it would have been more difficult if I were paired with a male instead. Some men like to monopolize a project like this.*

Also, some males make assumptions about your level before they get to see first hand so I was happy to get paired with a female".

"My first partner was a male who I previously knew and was friendly with, and though I felt comfortable with him, he made one mistake and I insisted it was a mistake and he insisted it wasn't, so I went along with him, and upon compile time I was correct. I feel my current partner, would have been more apt to take my advice, so maybe gender plays a role".

"Gender is somewhat a factor. Since we are both females, it was easier to talk to and trust. If my partner was of the opposite sex, it may cause conflict. I may not be able to voice my suggestions or be too distracted".

"I think gender probably would play a role in this experiment and I felt more comfortable programming with another female".

But interestingly, on the contrary, there were no similar comments found from the male subjects from [MM].

The following are the comments from the female partners from [MF]:

"my partner was male, and tended to not explain what he was thinking or doing which was sometimes frustrating."

"I think gender caused us to be a little more reserved around each other. If I programmed with a female I probably would have been more open."

"I think men in general tend to view women as inferior in the technological arena, whether it is intentional or unintentional. I, myself, have had to prove myself on many occasions to my male counterparts. It is when the grades are announced that they "accept" me, so to speak, as one of "them." I think this may have contributed to my partner's distrust. If I were male, I do think he would felt as though I was incompetent, but just may have thought that I was having a bad day or really needed a book for assistance. However, he made me feel as though even if I had a book, I still may not have known what to do. Of course this is speculation, and I do not think he meant it intentionally, but, I felt, his thoughts presented themselves unconsciously in his actions."

Based on the collected comments, [FF] shows a very strong level of affinity to each other. However, this appears to take away their ability to yield a better design and produce a higher productivity in situations such as PP.

Another interesting observation of this experiment is the quantitative measurement of one-man programming output versus pair programming output. From an overall assessment, the Myers's view (Myers and Myers, 1995) of ".... their shared preferences gives them common ground and their dissimilar preference gives them, as a team, a wider range of insights than either has alone" stands up very well to this experiment's findings. The key phrase is "a wider range of insights."

In the grand result, both code productivity (quantitative measurement) and code design (qualitative measurement) mean values were higher in PP teams than in one-man programming teams. The mean code design value showed no correlation between one-man programming and PP, and also exhibited a significantly higher PP value than in one-man programming. This infers that a PP team of two individuals with diverse MBTI personality types performs significantly better than two individuals in qualitative aspects of programming such as code design, code efficiency, and code readability. This grand result, again, supports the Myers's view.

Upon evaluating the results of three separate groups, opposite type, diverse type, and alike type, the results generally revealed that the mean values of both code productivity and code design yielded in a PP environment are higher than those obtained in a one-man programming environment.

One significant observation is the fact that the diverse type group's mean values of both code productivity and code design showed largest differences between PP teams and one-man programming teams. This further supports the Myers's view.

CHAPTER 5

CONCLUDING REMARKS AND FUTURE WORK

5.1 Summary of Findings

The following is a summary of experimental data and conclusions based on results from this study:

- The diversely MBTI type matched pairs outperformed the oppositely MBTI type matched pairs by 18% in productivity and by 19% in design.

- The diversely MBTI type matched pairs outperformed the similarly MBTI type matched pairs by 28% in both productivity and design.

- There are no significant differences between all three differently MBTI type matched groups in the levels of communication, satisfaction, confidence and compatibility

- The pairs that are diversely MBTI type paired and also male-male paired showed to be the best performing (productivity and design) teams.

- The pairs that exhibited a high level of communication between partners, did not necessarily experience a high level of satisfaction or exhibit compatibility between partners, nor did they have a high level of confidence regarding the finished product.

- Overall, female-female pairs tended to produce a lower output when compared to the male-male or male-female pairs.

- The communication skill level of each partner did not have any impact on the pair's performance with regards to design, productivity, satisfaction, confidence, and compatibility. The communication skill level seemed to have an impact on communication only.

- The gender factor had no impact on the pair's performance in design and productivity, however the level of communication, satisfaction and compatibility of male-male and female-female pairs was significantly higher than male-female pairs, with a much higher degree being exhibited by the female-female pairs.

- The PP teams generally have reported higher mean values of code productivity and code design than one-man teams.

- The PP teams of the diversely MBTI type matched pairs show significantly higher level of quantitative and qualitative coding work than one-man teams of diversely MBTI type matched pairs.

There were no interaction effects between MBTI type and gender, MBTI type and communication skill level, and communication skill level and gender.

5.2 Theoretical and Practical Implications

A list of theoretical and practical implications can be derived from the results of the experiment. Theoretical implications include the notion that MBTI personality type has a significant impact on PP. Different combinations of MBTI personality type clearly illustrate different levels of influences on PP as evidence in the areas of code productivity and code design. When examining the gender factor, same gender pairs show a higher level of communication, satisfaction, and compatibility while engaged in PP, with female and female pairs showing the strongest response. Other implications are that confidence on a job completed is not correlated to communication skill level, satisfaction, compatibility, MBTI personality type, or gender. Lastly, given the fact that measuring one's communication skill level is a difficult task, the instrument that was used to measure this attribute proved to be an effective and valid one. Many practical implications from this experiment can also be seen. One notable suggestion is for programming shop managers to pair the programmers according to one's personality profile, or MBTI personality type. Also a higher level of communication, satisfaction, and compatibility between the programmers can be achieved through same gender pairing. Additionally, it can be inferred that an appropriate level of managerial intervention and support in the pairing process is needed to alleviate the very fast and demanding pace of PP and also minimize any non-work related motives such as personal favoritism or office politics.

5.3 Limitations and Recommendations

One visible limitation with this experiment lied in the result analysis, whereby parametric tests were not used due to abnormal data distribution. This has limited the use of fine probing and analysis techniques such as multivariate analysis of variance (MANOVA). Where there are two or more dependent variables and also highly correlated to each other, as was the case in this experiment, it is highly recommended to use MANOVA (Niedrich, et al., 2001; Novak, 1995; Peter, et al.; 1975). This is because MANOVA is sensitive not only to mean differences but also to the direction and size of correlations among the dependents. Also due to this, no compounding variables such as grade point average (GPA) were tested. It's tenable that GPA may have had an influence on the pairs' outcome.

Another experimental limitation was that the number of female subjects may be not adequate enough to represent a statistically acceptable size. It was very difficult to allocate the size while still adhering to the testing hypotheses. For example, the number of female-female pairs for similarly MBTI type matched and same gender pairs in this experiment were only two. In general, locating an ample number of female students enrolled in software or programming courses or having computer science as a major may be difficult in an undergraduate level, although it tends to be less of a challenge at the graduate level.

Within this experiment, two specific recommendations are suggested. One is to expand this experiment to a longitudinal study. A semester long study (Muller and Tichy, 2001, Williams, 2000) would reveal more underlying psychosocial phenomena or maybe even contrasts some of the results of this experiment. Increasing the duration of the PP projects would simulate real-world projects whereby a typical project may last months or

years. It is during this extended period that various and unforeseeable psychosocial attributes and saliencies of cognitive interaction between two individuals may arise and thus provide a more true representation.

The other suggestion is to apply this experiment's findings to a group with more than just two individuals. Beyond the programming context, there are other cognitive-intensive tasks that would benefit from this experiment's findings (Deek and McHugh, 2003; Kwon, et al., 2002; Deek and McHugh, 2000; Wright and Cockburn, 2002). Ideally, a team of four to six persons working on a software development project could be experimented on to validate this experiment's findings (DeFranco-Tommarello and Deek, 2003; DeFranco-Tommarello, et al., 2003).

For agile software process paradigms, they will continue to develop, optimize, refine and evolve; the following areas deserve further research:

- Long-term validation
- Economical value (cost, benefits and tradeoffs)
- Dynamics of the underlying factors (a new business model?)
- Expansion to a distributed environment
- Changes and adaptations of existing tools
- Software Process Paradigm Framework
- Management issues

Long-term validation - As many of the agile paradigms are fairly new in their applications, the results are continuously observed and evaluated. The concept and framework are sound, but their practical sturdiness is to be observed (Abrahamsson, et

al., 2003). To skeptics, the long-term validation would allow them to gradually accept and adapt.

Economical value - Maybe the most expected point in using the agile paradigms is the economical value. If one values the time as priceless, then the fast and quick processes of the agile paradigms are unquestionably the choice. However, there is more involved in a software development process than just the time factor. A traditional model user may need to understand not only what he's getting, but also what he'll lose, such as a set of descriptively written software documents. Understanding the cost, benefit, and tradeoffs between the traditional models and the agile paradigms would allow an organization to better accept and migrate appropriately to one of the agile paradigms (Erdogmus, et al., 2002). The exchange also occurs among the stages of a model. The conventional order of the stages are rearranged or recreated in the agile paradigms. These economical perspectives are not only based on the economical values, but also on the efficiency of the process as well.

Dynamics of the underlying factors – Besides the explicit characteristics of the agile paradigms, there are the driving factors which makes the agile paradigms what they are and these factors ultimately touch on other relevant business issues and one may need to reevaluate the business process as well. If a business process has been embodying a linear sequential model then it certainly needs a new business process if XP replaces the linear sequential model.

Expansion to a distributed environment – Witnessing the exponential growth of globally dispersed software development (Carmel, 1997; Carmel and Agarwal, 2001), one may wonder how this concept will play a role in agile software process paradigms. In PP, virtual PP, a flavor of PP whereby two individuals, each from different

geographical location, virtually pair program using collaborative tools, is the subject of current experimentation. The question can be raised as to whether this concept of "virtuality" can apply to other agile paradigms. The future business ecosystem will inevitably shift faster and faster to the distributed environment.

Changes and adaptations of existing tools – Inevitably many existing practices and tools would have to change or evolve to fit into the agile paradigms. Being able to work without any upfront design documents, being able to accept last minute requirements, and having to deal with a fixed development time are all challenges that lead to change and adapt to new ways.

A framework for comparing and contrasting paradigms – A framework that provides bases and measurements for both traditional and non-traditional software process paradigms is expected. This framework compares and contrasts both traditional and non-traditional paradigms by common variables. The findings would reveal each paradigm's effectiveness and efficiency such as quality and productivity. This would allow one to estimate or forecast appropriate business application and usage for each paradigm. Also, the framework can determine whether the common variables or characteristics are present or not in a paradigm.

Management issues – From the survey results, there were voices of concern pertaining to the refining of PP in terms of appropriate management. The mental fatigue and "fair" pairing process have surfaced as issues to deal with. The demanding paired situation brings mental fatigue, causing deterioration in code quality and the pair relationship. The "fair" pairing process is also required to supercede any programmer's personal pair preference or objectives. For these and for other management issues, an assertive management's role or involvement must be studied and implemented.

Although the proposed future works are, in a broad sense, relevant to each other, each proposal requires distinct domain knowledge in order to provide a finer analysis. It is the authors' desire to analyze a majority of the proposals. However, considering the required time and resources, a number of collaborated researches with other interested researchers is expected.

For the immediate future, the authors will attempt to investigate two selected proposed future works: 1) expand this experiment's result into a longitudinal (semester long) study for the purpose of revealing more underlying psychosocial phenomena, and 2) integrate this experiment's result into a software development project team formulation process. As is with most team "jelling" processes, each member of a team is inclined to take up a particular role such as team leader or team facilitator. This natural phenomenon can be attributed to many factors such as one's past project experience or one's expertise in a particular field. Also, it can be attributed to intrinsic attributes, such as one's MBTI profile, as well (Yourdon, 1997). Based on the experimental findings of these two proposed future works, a substantial understanding can be acquired as to: 1) the impacts of psychosocial factors on software project team performance, and 2) the strategy of software project team forming.

5.4 Concluding Remarks

The software development community is at a crossroads today. The question that they are pondering over is whether to join the "agile" group or stay with the "established" methods of software development (Abrahamsson, et al., 2003; Boehm and Turner, 2003; Kent and Boehm, 2003). Despite the fact that the agile paradigms embody some anecdotal and forbidden practices, there is evidence of promising results. With the "no baggage" approach, the agile paradigms have given a new vehicle to react to the "Internet speed." The time factor has evolved into the most critical factor, followed by the increased software functionalities.

Through this research work, a visit to the process of software development is made, including a look at traditional development paradigms as well as some contemporary paradigms. Through this and other related research, a greater importance and significance of the human role are found in today's contemporary software development paradigms (Paulk, et al., 1993; Constantine, 1995; Yourdon, 1997; DeMarco and Lister, 1999; Cockburn, 2001; Cockburn and Highsmith, 2001). This may be due to the fact that humans can easily and instantaneously shift, change, and manage multi-tasks whereas a set of pre-defined procedures or protocols may actually impede the creativity and also the speed of delivering goods to an internet-driven market.

This notion of human importance aligns itself well with the spirit of the 'agile movement' of contemporary paradigms. PP is a fine example of how much of an impact the human factor can bring to the overall outcome. Each individual is different not only in their extrinsic attributes such as professional experience, programming skill or education, but also in their intrinsic attributes such as personality, cognitive process, and problem solving approach (Myers and Myers, 1995; Bayne, 1995; Deek, et al., 2000;

Deek and McHugh, 2000). It's common to have a pair of programmers who are exactly alike in their extrinsic attributes but completely different with regards to their intrinsic attributes. And this is what's being overlooked. It is the author's strong desire that many software shop managers who employ XP or PP benefit from this experiment's findings.

APPENDIX A

SUBJECT CONSENT FORM

NEW JERSEY INSTITUTE OF TECHNOLOGY
323 MARTIN LUTHER KING BLVD.
NEWARK, NJ 07102

CONSENT TO PARTICIPATE IN A RESEARCH STUDY

TITLE OF STUDY: Discovery and Analysis of Influencing Factors of Pair Programming

RESEARCH STUDY:

I,_____, have been asked to participate in a research study under the direction of Kyungsub Steve Choi and other professional persons who work with them as study staff may assist to act for them.

PURPOSE:

To better understand the impacts of factors such as personality, communication skill level and gender in pair programming.

DURATION:

My participation in this study will last for one semester.

PROCEDURES:

I have been told that, during the course of this study, the following will occur:

a. Attend the introduction information session and read and sign the subject consent form
b. Complete Myers-Briggs Type Indicator (MBTI) online personality assessment.
c. Engage in discussions with another subject for measuring communication skill level and complete Conversational Skills Rating Skill (CSRS) form.
d. Attempt sets of programming problems alone for two sessions and in a pair for three sessions (one pair session is a training session).
e. Complete the post session questionnaire.

PARTICIPANTS:

I will be one of about 150 participants to participate in this trial.

EXCLUSIONS:

I will inform the researcher if any of the following apply to me: None

RISK/DISCOMFORTS:

I have been told that the study described above may involve the following risks and/or discomforts:

NO risks and/or discomforts involved.

I fully recognize that there are risks that I may be exposed to by volunteering in this study which are inherent in participating in any study; I understand that I am not covered by NJIT's insurance policy for any injury or loss I might sustain in the course of participating in the study.

CONFIDENTIALITY:

Every effort will be made to maintain the confidentiality of my study records. Officials of NJIT will be allowed to inspect sections of my research records related to this study. If the findings from the study are published, I will not be identified by name. My identity will remain confidential unless disclosure is required by law.

PAYMENT FOR PARTICIPATION:

I have been told that I will receive $ 0 compensation for my participation in this study.

RIGHT TO REFUSE OR WITHDRAW:

I understand that my participation is voluntary and I may refuse to participate, or may discontinue my participation at any time with no adverse consequence. I also understand that the investigator has the right to withdraw me from the study at any time.

INDIVIDUAL TO CONTACT:

If I have any questions about my treatment or research procedures that I discuss them with the principal investigator. If I have any addition questions about my rights as a research subject, I may contact:

Richard Greene, M.D., Ph.D., Chair, IRB (973) 596-3281

SIGNATURE OF PARTICIPANT

I have read this entire form, or it has been read to me, and I understand it completely. All of my questions regarding this form or this study have been answered to my complete satisfaction. I agree to participate in this research study.

Subject: Name:_____

Signature:_____

Date:_____

SIGNATURE OF READER/TRANSLATOR IF THE PARTICIPANT DOES NOT READ ENGLISH WELL

The person who has signed above, _____, does not read English well, I read English well and am fluent in (name of the language) _____, a language the subject understands well. I have translated for the subject the entire content of this form. To the best of my knowledge, the participant understands the content of this form and has had an opportunity to ask questions regarding the consent form and the study, and these questions have been answered to the complete satisfaction of the participant (his/her parent/legal guardian).

Reader/
Translator: Name: _____
Signature:_____
Date: _____

SIGNATURE OF INVESTIGATOR OR RESPONSIBLE INDIVIDUAL

To the best of my knowledge, the participant, _____, has understood the entire content of the above consent form, and comprehends the study. The participants and those of his/her parent/legal guardian have been accurately answered to his/her/their complete satisfaction.

Investigator's Name: _____

Signature: _____

Date: _____

APPENDIX B

SUBJECT PROGRAMMING BACKGROUND INFORMATION

This form is used to collect the programming experience and background from the subjects.

Your Name:

Your ID:

School Year:

Major:

Your current programming course:

List all programming courses that you have taken (college level)

List all your professional programming experiences (Jobs, How long)

Have you practiced extreme programming before? If so, where and how long?

Have you practiced pair programming before? If so, where and how long?

APPENDIX C

POST PROGRAMMING SESSION QUESTIONNAIRE

This questionnaire was given to the subjects at the end of each visit to evaluate their programming experience.

Your Name:	Your course#:
Your ID:	Partner's ID:

There are total 43 questions. READ CAREFULLY, choose your choice correctly and then enter "X" in the box.

Reflect on your experience while you were PROGRAMMING ALONE and answer the following questions:

1. I am confident that work (answer) is done correctly.

Strongly Disagree			Not sure			Strongly Agree
1	2	3	4	5	6	7

2. My code is readable that others can follow and understand with no problem

Strongly Disagree			Not sure			Strongly Agree
1	2	3	4	5	6	7

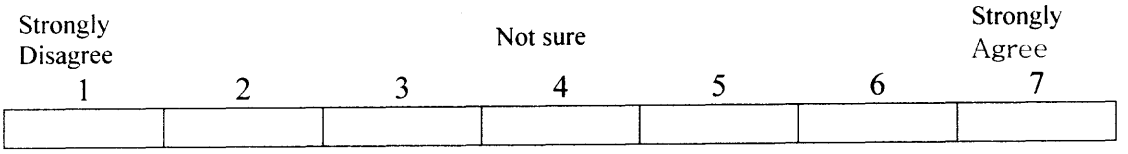
3. My code is efficiently written (with less number of lines).

Strongly Disagree			Not sure			Strongly Agree
1	2	3	4	5	6	7

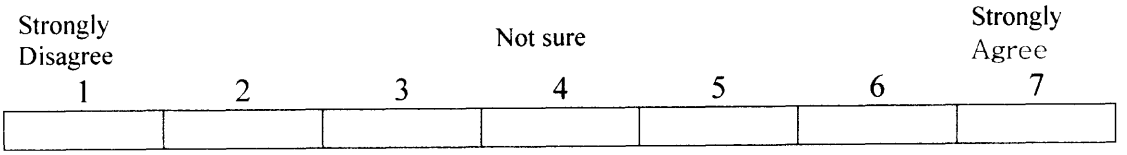
4. Compared to PP, I enjoyed the one-man programming session more

Strongly Disagree			Not sure			Strongly Agree
1	2	3	4	5	6	7

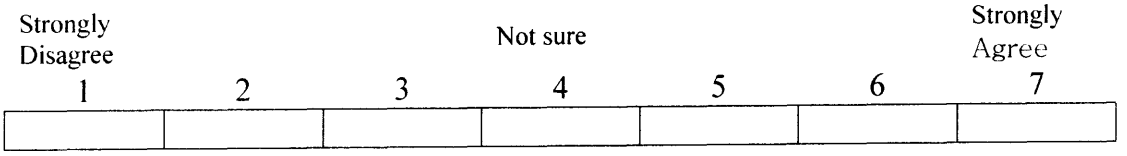
5. Compare to PP, I was NOT being creative in coding and generating the solutions



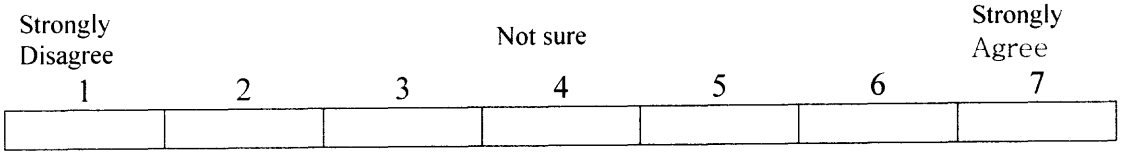
6. I program best when I'm left alone



7. I feel that I accomplish more when programming alone

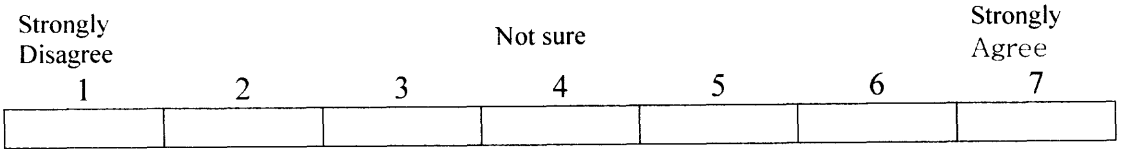


8. Compared to PP, I was able to produce a better solution when I programmed alone.

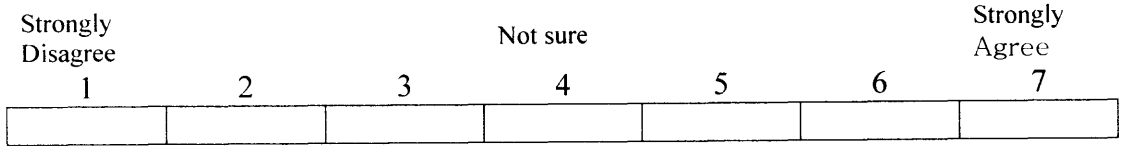


Reflect on your experience while you were PAIR PROGRAMMING (PP) and answer the following questions:*

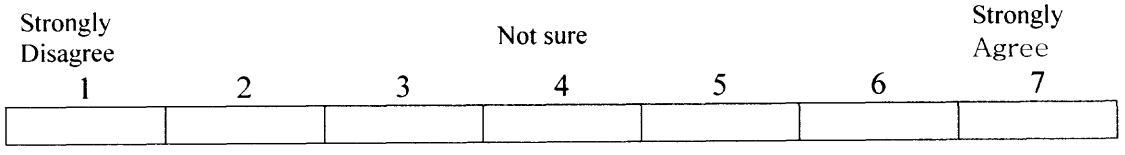
9. I am confident that our work (output) is done correctly.



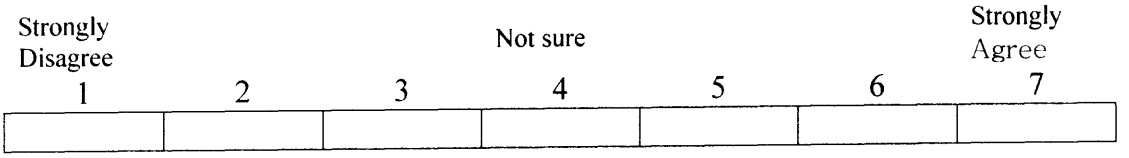
10. I believe that our code is readable that others can follow and understand it with no problem



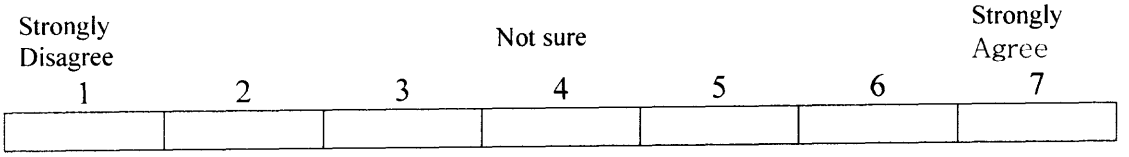
11. I believe that our code is efficiently written (with less number of lines).



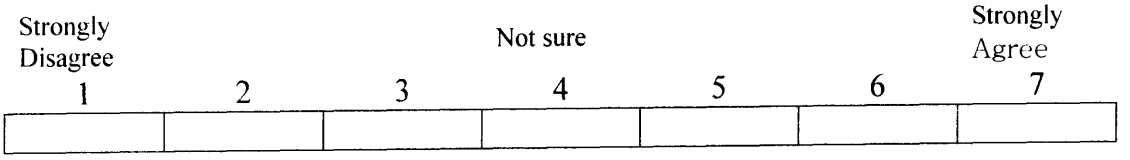
12. I enjoyed the PP session more than programming alone



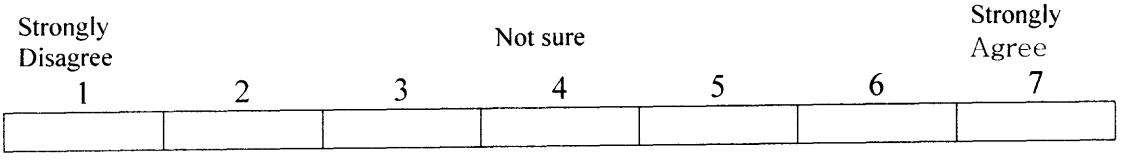
13. In PP, My partner and I disagreed frequently in reaching a solution.



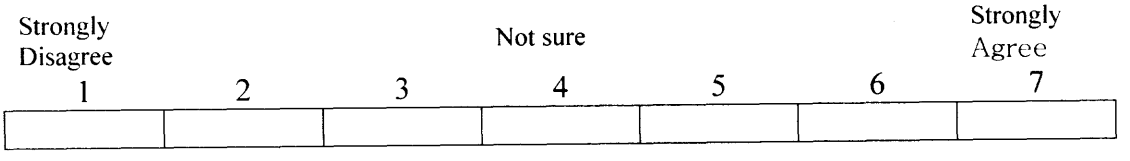
14. In PP, I believe that I had an easier time reaching the solutions.



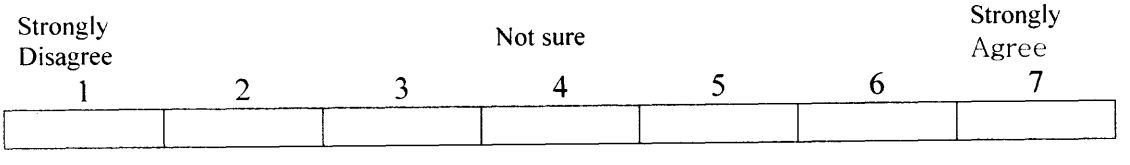
15. In PP I believe that we had much longer time reaching the solutions.



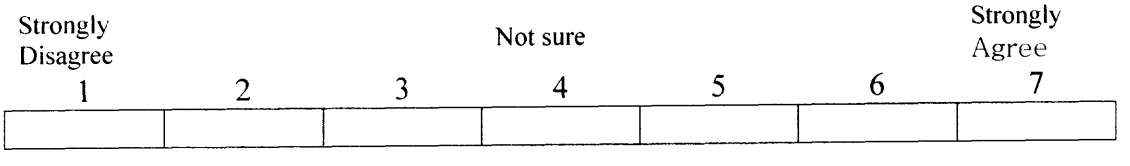
16. In PP, I did not like my partner's suggestion or view but I compromised to do (or follow) my partner's way.



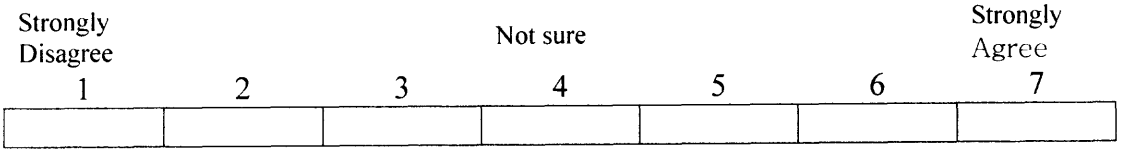
17. In PP, the feeling of trusting my partner grew stronger as the session went on



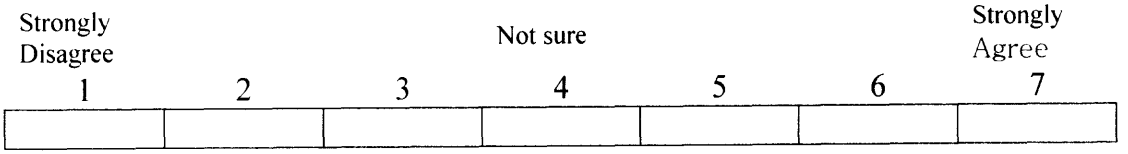
18. My partner showed things that I did not know about



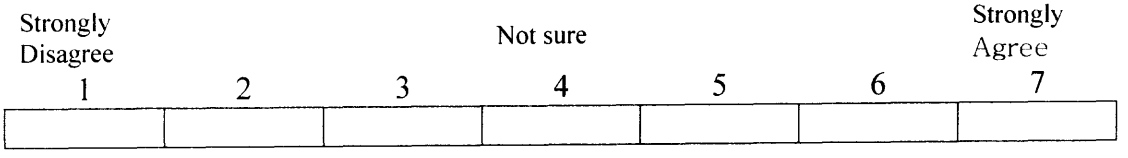
19. In PP, I really enjoyed my partner's partnership (willing to work as a team, open mind, etc)



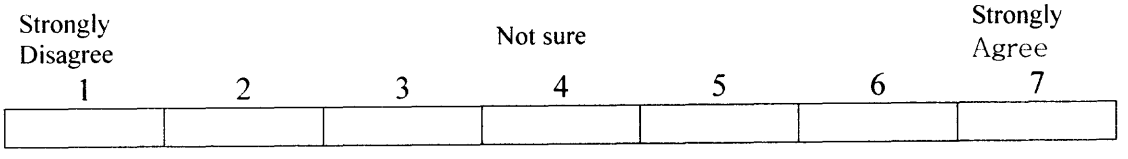
20. In PP, my partner insisted in doing things his way and/or did not collaborate



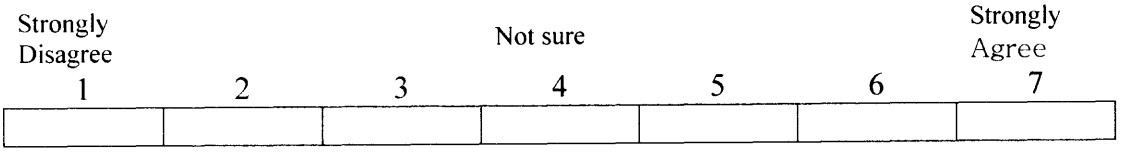
21. In PP, I felt that I was NOT able to fully exercise my programming skill



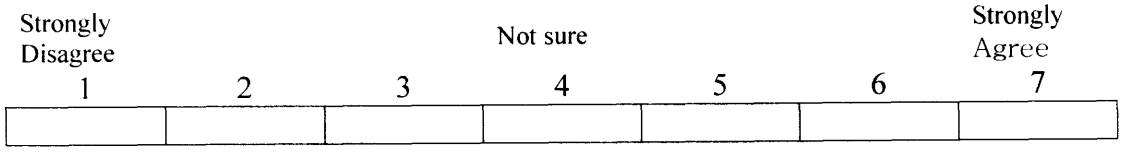
22. In PP, I believe that I'm a compatible partner to my partner



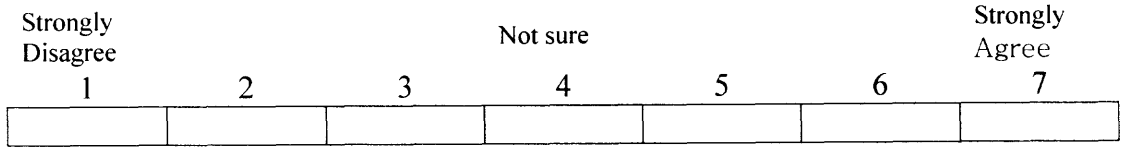
23. In PP, It was my politeness and etiquette that got through the PP sessions, NOT my true self (personality)



24. IN PP, I BELIEVE THAT I HAVE A COMPATIBLE PERSONALITY TO MY PARTNER



25. IN PP, I DID NOT REVEAL MY TRUE SELF, PERSONALITY DURING THE WHOLE SESSION



26. In PP, there were times when I was withdrawn (or maybe upset) because of the disagreements from the partner

Strongly Disagree			Not sure			Strongly Agree
1	2	3	4	5	6	7

27. I think PP is a form of cheating as PP takes away the opportunity of one to truly learn on his or her own.

Strongly Disagree			Not sure			Strongly Agree
1	2	3	4	5	6	7

28. I don't think I can achieve a higher productivity next time if I'm paired with my partner again

Strongly Disagree			Not sure			Strongly Agree
1	2	3	4	5	6	7

29. In PP, my partner described his or her point very well and I was able to fully understand

Strongly Disagree			Not sure			Strongly Agree
1	2	3	4	5	6	7

30. In PP, my partner did not express nor communicated much (too quiet) which made PP very difficult

Strongly Disagree			Not sure			Strongly Agree
1	2	3	4	5	6	7

31. In PP, my partner's message delivery was unclear which made PP very difficult

Strongly Disagree			Not sure			Strongly Agree
1	2	3	4	5	6	7

32. In PP, my partner's active communication to me allowed me to be more active in expressing my views as well.

Strongly Disagree			Not sure			Strongly Agree
1	2	3	4	5	6	7

33. In PP, my partner's voice tone was loud and clear which helped our communication.

Strongly Disagree			Not sure			Strongly Agree
1	2	3	4	5	6	7

34. In PP, my partner's hand gestures, eye gaze, body positions and other communication cues were NOT used adequately and poorly managed in our communication.

Strongly Disagree			Not sure			Strongly Agree
1	2	3	4	5	6	7

35. In PP, the gender of my partner did not influence my PP experience at all

Strongly Disagree			Not sure			Strongly Agree
1	2	3	4	5	6	7

36. In PP, the fact that my partner was male/female it allowed me to focus more on the problems

Strongly Disagree			Not sure			Strongly Agree
1	2	3	4	5	6	7

37. In PP, I think the gender can be an issue

Strongly Disagree			Not sure			Strongly Agree
1	2	3	4	5	6	7

38. Describe any conflicts or negative impacts from your partner that you have experienced during the pair programming which you think it (or they) influenced or caused the programming productivity. (Talk about what specifically happened in YOUR PP session)

39. Describe any positive impacts from your partner that you have experienced during the pair programming which you think it (or they) influenced or caused the programming productivity. (Talk about what specifically happened in YOUR PP session)

40. Discuss the compatibility (personality, communication or others) of you and your partner. If good, why? If not, why not? Do you think you can achieve a high productivity with your partner again? (Talk about what specifically happened in YOUR PP session)

41. Besides your partner's programming skill, How did the personality factor played a role in your and your partner's collaboration? Was the personality compatible or NOT? Describe. (Talk about what specifically happened in YOUR PP session)

42. Besides your partner's programming skill, How did the communication skill factor played a role in your and your partner's collaboration? The level of communication skill that your partner exhibited, Did that bring on a positive (or negative) feeling and impact to your collaboration back to your partner? Describe. (Talk about what specifically happened in YOUR PP session)

43. Besides your partner's programming skill, How did the gender factor played a role in your and your partner's collaboration? Your partner being male or female, Did that influence to your collaboration back to your partner? Do you think that you would had a different pair programming experience if your partner was the opposite sex? Describe. (Talk about what specifically happened in YOUR PP session)

APPENDIX D

PAIR PROGRAMMING EXPERIMENT FIRST VISIT SCHEDULE

These were the items that were covered during the 1st visit by the subjects.

1) PP background

   - Give an informal talk to introduce and present PP concept and practice.

2) Subject consent form

   - Go over the form and discuss any issues. Obtain the signatures and collect the form

3) Comm. skill eval

   - Give a background on level of communication skill measurement
   - Pair up two students and give the first topic (appendix F)
   - Give a 20 minutes for the pair discussion
   - After the pair discussion, briefly hold a class discussion
   - Give the second topic (appendix G)
   - Give a 20 minutes for the pair discussion
   - After the pair discussion, briefly hold a class discussion
   - Split up the pairs and have a partner sit apart from his or her partner
   - Give the CRCS form (appendix E)
   - After the forms are filled out, collect them.

4) MBTI personality assessment

   - Give a background on MBTI and its objective in this experiment
   - Assist each subject to take the online MBTI assessment
   - During the assessment, assist the subjects with any requests or questions

5) PP session

   - Ask the subjects to pair up and find a terminal
   - Remind the subjects about the PP concept and the purpose of this training session
   - Give a set of simple problems from their textbook
   - After each subject had a few opportunities to drive the keyboard and complete a few problems, they are to be told that they may stop and leave
   - The subjects are reminded of the second visit information

APPENDIX E

COMMUNICATION SKILL LEVEL MEASURING INSTRUMENT I

This is Conversational Skills Rating Scale (CSRS) form that is used by the subjects to judge his or her partner's level of communication skill.

Your ID:	Partner's ID:

Rate your partner according to how skillfully he or she used, or didn't use, the following communicative behaviors in the conversation, where:

1 = INADEQUATE -use was awkward, disruptive or resulted in a negative impression communicative skills
2 = SOMEWHAT INADEQUATE -occasionally awkward or disruptive, occasionally adequate
3 = ADEQUATE -use was sufficient but neither very noticeable nor excellent. produced neither positive or negative impression
4 = GOOD -use was better than adequate but not outstanding
5 = EXCELLENT -use was smooth, controlled, and resulted in positive impression of communicative skills

Circle the single most accurate response for each behavior:

1 2 3 4 5	1 Speaking rate (neither too slow nor too fast)
1 2 3 4 5	2 Speaking fluency (avoided pauses, silences, "uh", etc.)
1 2 3 4 5	3 Vocal confidence (neither tense nor nervous sounding)
1 2 3 4 5	4 Articulation (language clearly pronounced and understood)
1 2 3 4 5	5 Vocal variety (avoided monotone voice)
1 2 3 4 5	6 Volume (neither too soft nor too loud)
1 2 3 4 5	7 Posture (neither too closed/formal nor too open/informal)
1 2 3 4 5	8 Lean toward partner (neither too far forward nor too far back)
1 2 3 4 5	9 Shaking or nervous twitches (weren't noticeable)
1 2 3 4 5	10 Unmotivated movements or fidgeting (e.g., pencil, rings, hair, fingers, etc.)
1 2 3 4 5	11 Use of eye contact
1 2 3 4 5	12 Facial expressiveness (neither blank nor exaggerated)
1 2 3 4 5	13 Nodding of head in response to partner's statements
1 2 3 4 5	14 Use of gestures to emphasize what was being said
1 2 3 4 5	15 Smiling and/or laughing
1 2 3 4 5	16. Use of humor and/or stories
1 2 3 4 5	17. Asking questions
1 2 3 4 5	18. Encouragements or agreements (encouraged partner to talk)
1 2 3 4 5	19 Speaking about partner or partner's interests (involved partner as topic of conversation)
1 2 3 4 5	20. Speaking about self (didn't talk too much about self/own interests)
1 2 3 4 5	21. Expression of personal opinions (neither too passive or aggressive)
1 2 3 4 5	22. Initiation of new topics
1 2 3 4 5	23. Maintenance of topics and follow up comments
1 2 3 4 5	24. Interruption of partner's speaking turns
1 2 3 4 5	25. Use of time speaking relative to partner

For the next 5 items, rate the person's overall conversational performance:

(26) POOR CONVERSATIONALIST :1 2 3 4 5 6 7: EXCELLENT CONVERSATIONALIST

(27)	SOCIALLY UNSKILLED	:1	2	3	4	5	6	7:	SOCIALLY SKILLED
(28)	INCOMPETENT INTERACTANT	:1	2	3	4	5	6	7:	COMPETENT INTERACTANT
(29)	INAPPROPRIATE INTERACTANT	:1	2	3	4	5	6	7:	APPROPRIATE INTERACTANT
(30)	INEFFECTIVE INTERACTANT	:1	2	3	4	5	6	7:	EFFECTIVE INTERACTANT

Scoring for CSRS form is simply summing up the responses of each item. Following is a leaf-and-stem distribution of the pair programming experiment subjects' CSRS form total scores.

```
 80|
 85|7,9
 90|0,1,1,4
 95|9,9
100|
105|6,6,6,6,7
110|1,4,4
115|5,5,6,6,6,7,7,7,7,8,8,9,9,9
120|1,1,1,1,2,2,3,4,4,4,4
125|5,5,5,5,6,6,7,7,7,7,7,7,9,9,9,9
130|0,0,0,1,1,1,1,2,2,2,3,3,3,4,4,4
135|5,6,6,6,6,6,7,7,7,7,8,8,8,9,9,9
140|0,0,1,1,2,2,2,2,3,3,4,4,4,4
145|5,6,6,6,6,7,7,7,8,8,9,9,9,9,9
150|0,1,1,1,2,2,3,3,4,4,4,4
155|5,5,8,9,9
160|0
```

The subjects who have scored from 80 to 127 are labeled with "low" communication skill level (<40%), 128 to 137 are labeled with "medium," and 138 to 160 are labeled with "high" (>60%) This scoring and grouping manner was followed by the CSRC form inventor, Dr. Spitzberg (Spitzberg, 2002; Spitzberg, 1997; Spitzberg and Cupach, 1984)

APPENDIX F

COMMUNICATION SKILL LEVEL MEASURING INSTRUMENT II

This is one of two tasks that are given to the subjects for the purpose of measuring communication skill level.

Instruction: You and your partner are given a question (see below). After a few minutes of reflection, face your partner, engage in an active discussion, and arrive at a final answer that both you and your partner agree on. You are asked to fully present your views to your partner.

Question: New Jersey Institute of Technology (NJIT) is currently in talks of merger with other local major universities. As a student to your university how do you feel about this merger talk? Do you believe there are more benefits to gain for your university? If not, why not? How about for yourself?

APPENDIX G

COMMUNICATION SKILL LEVEL MEASURING INSTRUMENT III

This is one of two tasks that are given to the subjects for measuring communication skill level.

Individual Ranking: Read the Noble Industries case, Employee Profile, and Supervisor's Comments below and rank the employees per instruction.

Directions: Read the case below and discuss it with your partner in completing the ranking. Below you will find the names of the ten (10) employees who may be laid off. In the table below place a number from 1-10 in the column next to the name of each individual. Assign number 1 to the first person you think should be laid off, then number 2 and so on. The last person to be laid off should be assigned number 10. Ties are not permitted.

Employee Name	Barbara	Chris	Fred	Harry	Joanne	Lois	Phil	Sharon	Susan	Tom
Rank (1-10)										

NOBLE INDUSTRIES

Noble Industries is a mid-sized, diversified manufacturing firm with corporate headquarters located in Columbus, Ohio. The company was founded in 1958 and has experienced steady and continuous growth for most of its forty-year history. Eight manufacturing facilities are located in different parts of the United States and each plant employs approximately 250 people. Gross revenues for Noble Industries in 1997 were $105 million.

The Information Systems Division (ISD) at Noble Industries is functionally distributed throughout the organization. Each plant is responsible for developing and supporting its own local IS operations (for example, ordering, production scheduling, quality assurance, decision support, etc.). All corporate-wide systems (human resources, sales forecasting, research, executive information, etc.) are managed from the central Information Systems Division at corporate headquarters. A total of 150 people are employed in the Information Systems Division (ISD).

After graduating from college in 1988 you were hired as a junior programmer at the Columbus, Ohio site. Five years later you were promoted to the position of systems analyst. Now you are a senior systems analyst in ISD. Your group is responsible for application development and software support for the Research and Development

Division (RDD). There are two systems analysts, four applications programmers, and 2 clerical support staff who report directly to you.
This morning, you and the other senior systems analysts at corporate headquarters met with Bob Thompson, Vice President for Information Systems. He explained, "At yesterday's Executive Management meeting it was announced that some of our durable goods customers have found new suppliers. Based on the loss in sales, our V.P. for Finance has projected a 3-6% decline in gross revenue this year. The CEO said that unless sales increase very soon we will have to make staff reductions. All Division Vice Presidents have been asked to develop a preliminary list of people who would be laid off. For ISD, at least one and perhaps as many as ten staff members could be terminated. The final decision will be made next week."

Bob then distributed envelopes marked "Confidential". He said, "In each envelope you will find profile information on ten information systems employees. I want you to take the envelopes back to your office, read the profile and supervisor's comments, then rank the employees in the order that you think they should be laid off. The employees' real names don't appear on the profile page because I don't want you to have to make a decision about someone you know. As these are all good employees who have been performing well, management felt it wanted a rating by an impartial group of technical peers as input to their final decision. Use your best judgment. Then I want you to get together with your partner discuss your individual rankings, and submit a final ranking to me by the end of the day."

Bob concluded the meeting by saying, "I understand this is not an easy task, but given the current situation we don't have any other choice". At that point, you and the other senior systems analysts went back to your offices to work on the ranking assignment.

Employee Profile

Name	Age	Title	Years with Co.	Education (highest degree)	Education	Marital Status	# of Dep.
Barbara	27	Senior Systems Analyst	2	MBA	Stanford Univ.	Single	0
Chris	35	Systems Analyst	10	BS - MIS	U. of Oklahoma	Married	4
Fred	61	Systems Programmer	27	DP School	DeVry Tech.	Married	1
Harry	27	Applications Programmer	4	BS - CIS	Univ. of Bombay	Single	3
Joanne	46	Senior Systems Analyst	15	MS -MIS	Ohio State Univ.	Married	2
Lois	54	Database Administrator	22	AAS - DP	Miami Dade CC	Widowed	0
Phil	26	Systems Analyst	3	BS - CIS	Natl. Taiwan Univ.	Single	1
Sharon	36	Clerical	12	High Sch.	Harrison H.S.	Married	6
Susan	51	Applications Programmer	9	BS - CIS	Texas A&M Univ.	Divorced	3
Tom	47	Senior Systems Analyst	18	BS - Mgmt.	Purdue Univ.	Divorced	4

Supervisors' Comments

Barbara: "Barbara is very ambitious and always asks for the most challenging assignments. She believes that hard work should be recognized and rewarded. For example, at both annual performance evaluations Barbara has wanted to know when she will be considered for a promotion. She defines success in terms of position and salary. Barbara is very competitive and I suspect that she will move up in the management ranks. My only concern is that sometimes she can be too assertive."

Chris: "Of all the people in my department, Chris responds the quickest when I give him an assignment and he never asks why I want something done. Chris respects authority and understands that everyone needs to know their place in the organizational hierarchy. He doesn't mind bureaucracy because he knows it improves efficiency. Maybe that's why Chris enjoyed being in the army for eight years."

Fred: "Fred has been with us for a long time. He enjoys his work and is grateful for the job security he's had here all these years. Fred gets along with everyone and gets a lot of satisfaction out of helping others, especially some of the newer employees when they have questions. He recognizes that it's important for people who work together to agree on things. Fred doesn't like controversy, so he is willing to compromise when others disagree with him."

Harry: "Harry is the most technically competent guy in my department. He reads every technical report he can get his hands on. However, Harry likes doing things his own way and prefers to work alone. In this way, Harry believes that it will be easier for me, his supervisor, to reward him for the work he does without anyone else getting any credit. His priorities are very clear, he puts himself and his family above everything else in his life."

Joanne: "Joanne has excellent organizational skills. She understands that we need to have rules and the rules need to be followed. Joanne is the department's quality assurance leader (QAL) because she believes that order and structure are necessary for productivity. She doesn't take any unnecessary risks, and thoroughly researches something before making a recommendation to me."

Lois: "Lois is a real team player. She really enjoys working with others and always puts the group's interests ahead of her own. In fact, when the new database conversion was completed she suggested that the whole group be recognized for the achievement even though I knew that Lois did most of the work. I know that community service is also important to Lois. She is a volunteer at the local shelter for the homeless."

Phil: "Although he's only twenty-six years old, Phil tends to live his life as if he were much older. He likes to study history and the way people lived in the past. Last year when Phil's mother became ill he moved back home to tack care of her. Phil has a great respect for other people and will defend them when they are being criticized, whether the criticism is justified or not."

Sharon: "Sharon is very dedicated. She comes to work early, usually stays late, and always completes every assignment no matter how long it takes. Sharon places a great deal of emphasis on her relationships with others. Her long-term goal is to retire in Florida, so she tries to save as much money as she can."

Susan: "I have known Susan for about seven years, ever since she started working here. She gets her work done and thinks that everyone should do their fair share. In fact, Susan once told an assistant plant supervisor who was visiting our plant that he should try doing her job for a day. I give Susan a lot of credit, she speaks her mind and doesn't care if you're the CEO or the mail room clerk. To her, no one is any better than anybody else."

Tom: "Tom is always the first to try something new. It doesn't make any difference whether it's a radio station, a place to vacation, or a style of clothes. Tom does things his own way and he's not afraid to break the rules. With me, Tom is the same way. So I generally give him the assignments that have a big risk, but potentially a big payoff."

APPENDIX H

PROGRAMMING PROBLEM SET

Following four problems are used in the experiment.

Problem: Life Insurance

The insurance company offers three plans; Plan A - $100,000, Plan B - $500,000, Plan C - $ 1,000,000. For the Plan A the standard monthly premium (stdrt) is $10, plan B is $20, and plan C is $50.

Write a program, which calculates the health insurance monthly premium. The user inputs name, age, gender, medical history, marital status, smoking/non-smoking, exercise/no-exercise, years with the plan, availability to change option and whether it's new or renewal. The program outputs the customer's name, age, gender and the recalculated monthly premium.

If the customer has been with the plan for 5 or more years the customer has an option to change to a different plan at **every fifth year**. However, if the customer wants to change when he/she is on a year that is not a fifth year, he/she may do so with a penalty of 1.05 factor. For example, if it's fifth, tenth, fifteenth, etc...with the plan, he/she may change it w/o a penalty, but if he/she wants to change it on seventh, thirteenth, seventeenth, he/she may with a penalty (1.05). If the customer has been with the plan less than 5 years, this option is not available.

Plan B is the most profitable product, if an organization with 150 employees or more chooses Plan B for all their employees, then each employee receives a 0.95 factor.

The factors are cumulative, so if a customer is "37 years old, male, medical "high" risk, married, Exercises regularly, non-smoker" and also changing the plan with a penalty then the recalculated premium would be (1.2*1.0*1.5*0.8*0.7*1.0*1.05)*stdrt. For a condition that is not listed in the table use 1.0 (i.e. male, non-smoker)

Condition	Factor	Condition	Factor
Age <36	0.7	Married	0.8
35<age<48	1.2	Single	1.2
47<age<65	1.5	Smoker	1.5
64<age	1.7	Regular Exercising	0.7
Female	0.9		
Medical history "high" risk	1.5		
Medical history normal condition	1.0		
Medical history "low" risk	0.8		

Problem: Online Supermarket

Your team is asked to create a program that is to be used by an online supermarket. You need to observe the following;

1. Each item in the store has a standard price that is up to two decimal places in precision.
2. A state sales tax (7%) is charged on all items except: bottled water, toilet paper, and fresh fruit.
3. A customer must buy at least 5 items, which includes 2 or 3 promo items and one non-taxed item.
4. A customer may buy more than one quantity per item.
5. The extended price (unit price * qty) for an item shall never be less then zero.
6. The store may run a promotion where the customer can buy three for a special price. If the customer buys less than three while the promotion is in effect the price is equal to the special price divided by the quantity rounded down to two decimal places.

Items	Price ($)	Promotions?	Promo Price ($)
A loaf of bread	1.99		
A box of orange juice	5.99		
One pound of Apple	2.99		
A box of Ice cream	3.99		
A set of two rolls of Paper towel	1.99	Yes	3 for $4.99
A bottled water	0.99		
A bottle of Ketchup	4.99		
Pecan Pie	3.99		
A box of Laundry Detergent	8.99	Yes	3 for $14.99
A bottle of Olive Oil	2.99		
One bag of grapes	4.99		
A set of 12 rolls of toilet paper	9.99	Yes	3 for $16.99
A case of Tofu	1.99		
A bag of Imported black beans	2.99		

A shopper would take the printout and go to the supermarket to pay and pick up the grocery items.

Problem: Emergency Medic

During a time of war, the number of field surgeons and medic staff is always in a shortage. As a remedy to this, the department of defense is testing an idea where a junior medic soldier can be deployed. To help the junior medic soldier, a PDA containing special program is to be used. It is to give a minimal but life-saving emergency treatment on the spot to save the soldiers' lives. With the help of the PDA, the medic soldier assesses the wounded soldier and performs an appropriate treatment. As the lead developer, you are asked to create the program. For this pair programming experiment purpose, please use your common sense regarding to the uses and types of treatments.

The program asks the following questions and depending on the answers a treatment plan will be output.

- Type of wounds
  - abdominal wound
  - head wound
  - shattered bone
  - broken bone
  - burn injury
  - Injury from explosive devices fragments
  - Bio-Chemical injury

- Pulse rate
- Body temperature
- Conscious or unconscious
- Severe bleeding or not
- Experiencing difficulty breathing

Output will be the solider name, solider military ID, the list of all inputted entries, and the list of appropriate treatments.

Treatments

- Remove the large/noticeable fragment pieces
- Tightly wrap the wounded area with a roll of gauze
- Apply antibiotic cream
- Protect and shield the wounded area with a flexi-support beam
- Wash the wounded area
- Perform a pain-killing shot
- Perform an antibiotic shot
- Place a medic-band-aid
- Immediate transport to a field hospital required
- Elevate legs

As a medical note, following conditions applies;

1) Any head wounds and unconsciousness require an immediate transport to a field hospital as a final step
2) If the body temperature is above 105 F or below 90 F, do not perform the pain-killing shot.
3) Shattered bone injury receives 1.5 times dosage of painkiller.
4) Regular bleeding may get the medic-band-aid, but severe bleeding does not.
5) For bio-chemical injury, do not wash with water on the area.

Problem: Employee Pay

Write a program called Pay, which calculates the pay for an employee's. Assume the user inputs employee name, status (i.e., full/part time), number of hours worked and payrate on the command line in that order.

Your program should output the employee name, number of hours worked and pay.

When the employee worked 40 hours or less he/she should get regular pay

When the employee worked between 40 and 60 hours (from 41 to 60) and he/she is a full time employee he/she should get doubleovertime, if he/she is a partime employee then
he/she should get time and a half.

When the employee worked between 60 and 80 hours (from 61 to 80) and he/she is a fulltime employee he/she should get triple overtime, if he/she is partime then
he/she should get double overtime.

When he works more than 80 hours, nothing is calculated and your program should print an error message that states, "meet with employee".

APPENDIX I

PROBLEM EVALUTION FORM

Background:
Four programming problems are developed for the purpose of assessing the compatibility, level of collaboration, and teamwork in a pair programming context. The focus of the problems is not to challenge or test the subjects' limits with regards to their programming skills, but to provide enough of a challenge in code requirements and design so that a pair of subjects actively discusses and collaborates for the solutions to the problems. In order to achieve this in the four problems, it is important that all of the four problems present the same level of programming difficult. Three programming experts are asked to verify this.

I, _____ was asked by Kyungsub Steve Choi to review the four problems (problem names: life insurance, online supermarket, emergency medic, and employee pay) and give my judgment. After a careful review of the four problems I agree that the four problems present the same level of programming difficulty.

Evaluator Name (Print):

Evaluator Signature:

Reviewed Date:

APPENDIX J

INSTRUCTION SHEET FOR THE JUDGES

Instruction for the Judges:

There are two categories; quantitative measure (How much of coding was done?) and qualitative measure (How are the codes?). The scale is 0 –10, 10 being the highest score that a judge can give.

- Quantitative Measurement – Assign an appropriate score based on code productivity
    - Coding activity that shows the programmer(s) has attempted to solve the problem. However one or two lines are not acceptable.
    - Coding activity that shows the programmer(s) has completed almost half of the coding work
    - Coding activity that shows the programmer(s) has either completed all the coding work (including output) or close to completing the coding work
- Qualitative Measurement – Assess an appropriate score based on code design (structured, modular, or object-oriented approach), code efficiency, and code readability

*****Please contact the experiment investigator if this instruction is NOT clear*****

APPENDIX K

ILLUSTRATION OF TWO JUDGES'S SCORES

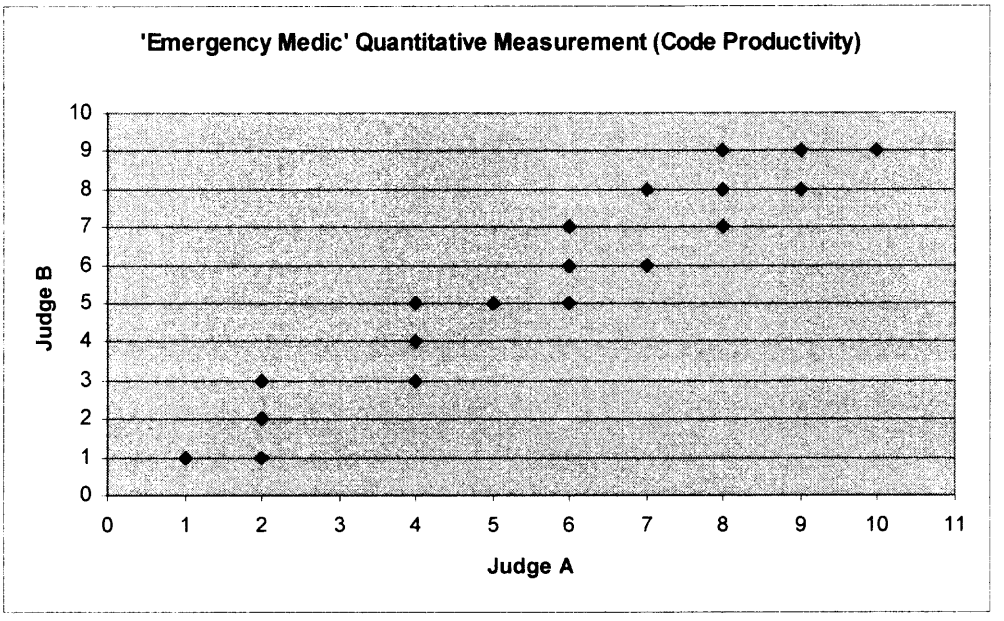


Figure K.1 'Emergency Medic' Quantitative Measurement (Code Productivity)

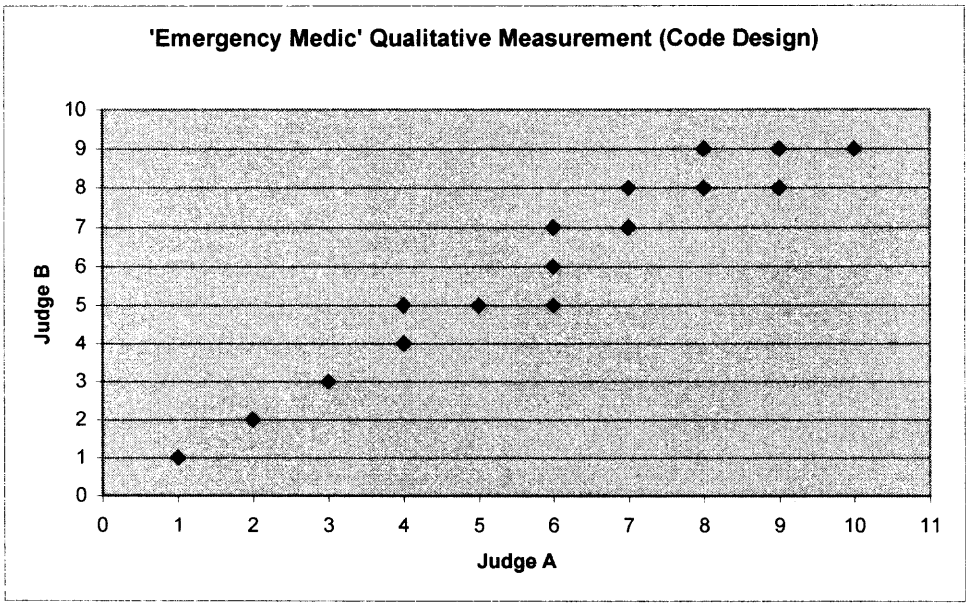


Figure K.2 'Emergency Medic' Qualitative Measurement (Code Design)

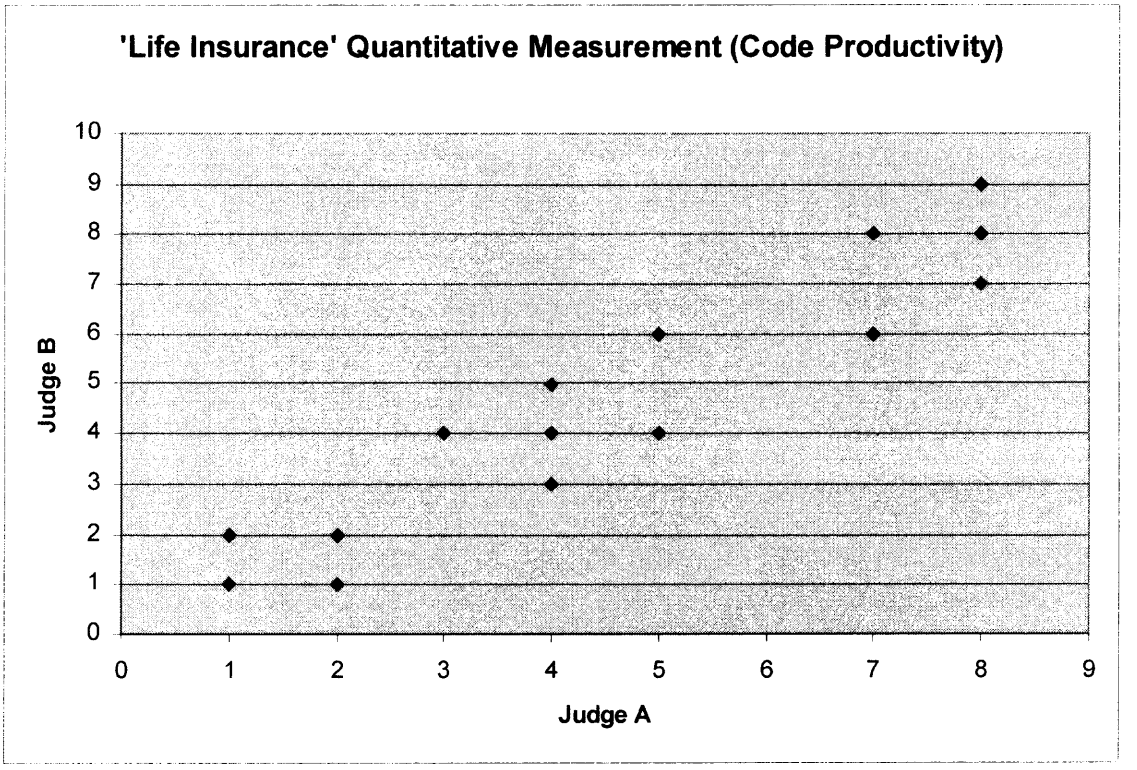


Figure K.3 'Life Insurance' Quantitative Measurement (Code Productivity)

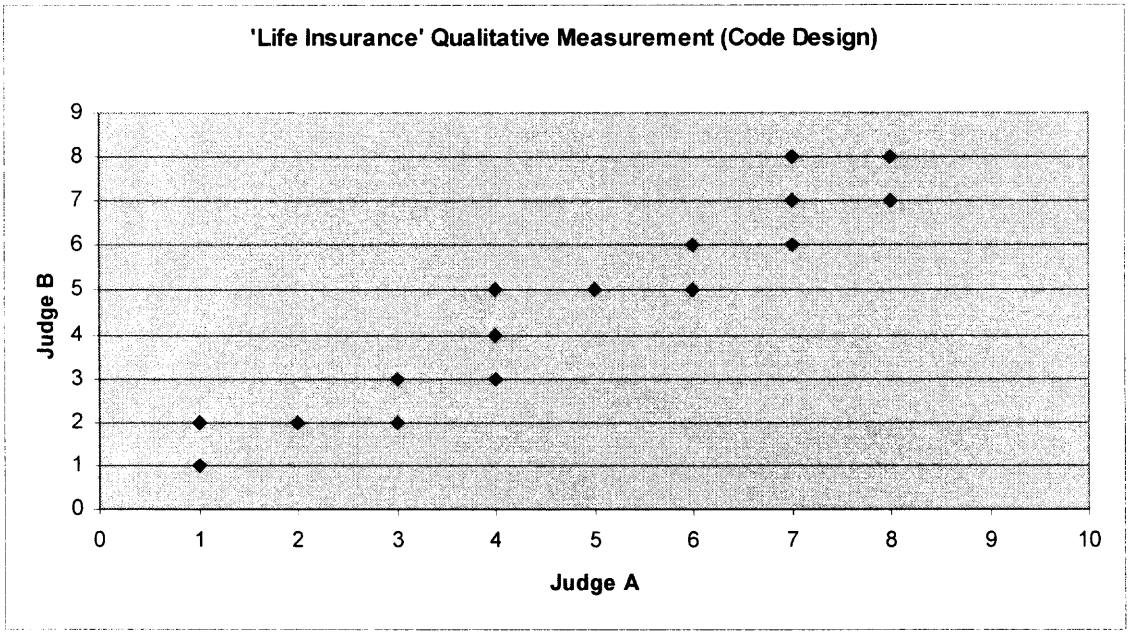


Figure K.4 'Life Insurance' Qualitative Measurement (Code Design)

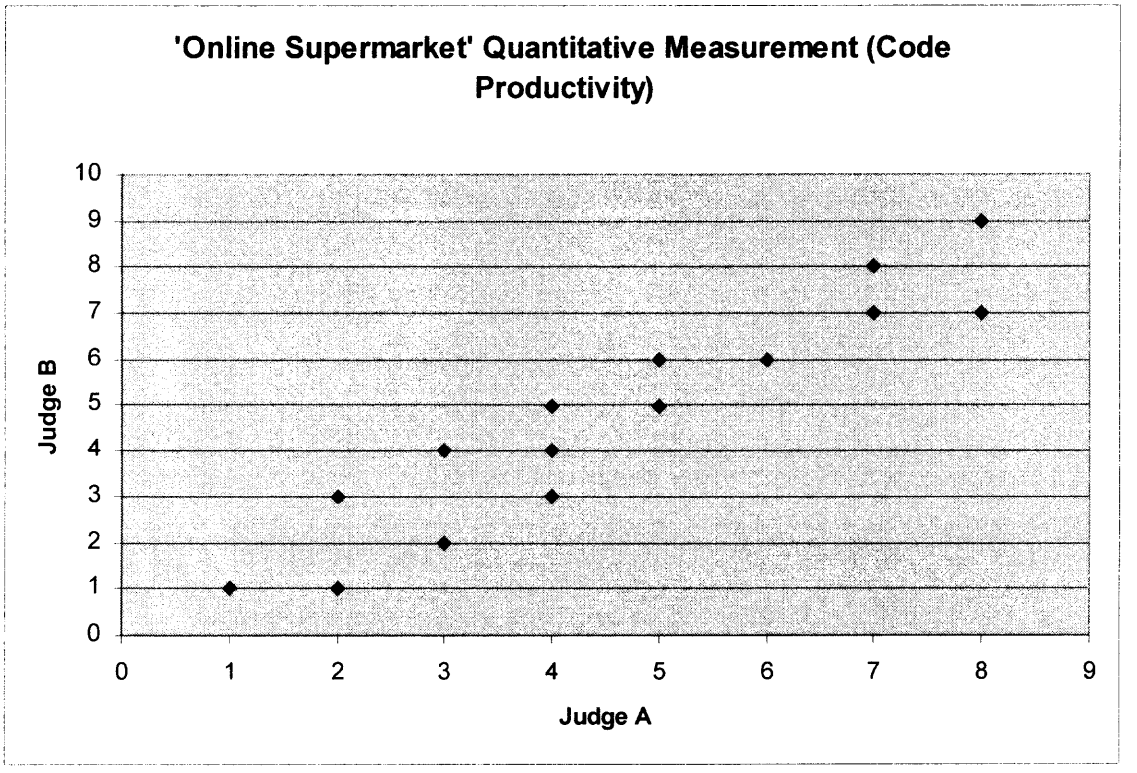


Figure K.5 'Online Supermarket' Quantitative Measurement (Code Productivity)

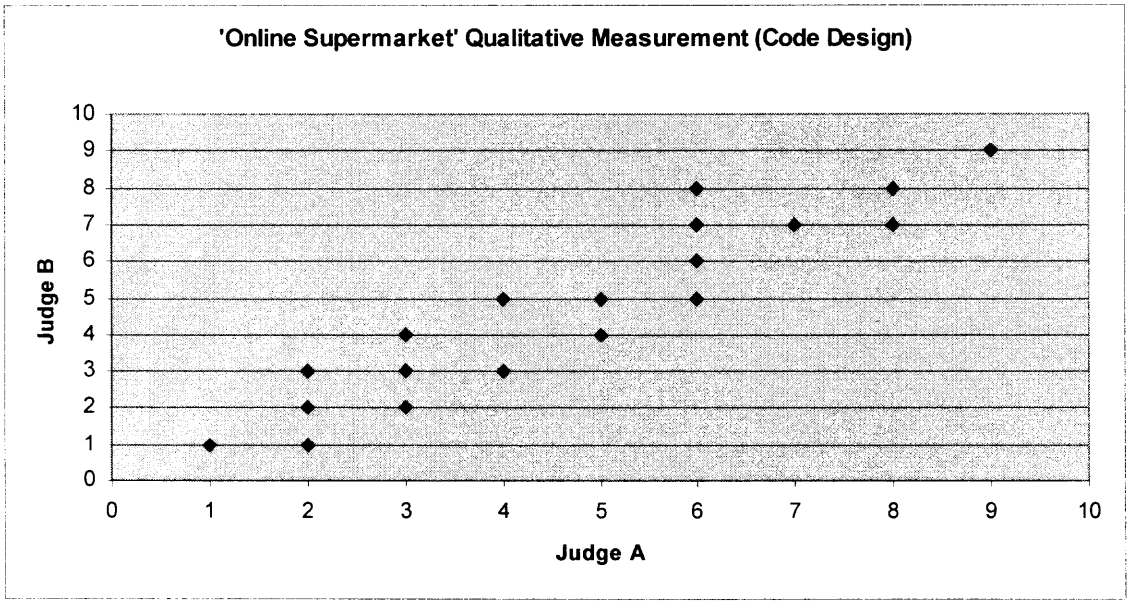


Figure K.6 'Online Supermarket' Qualitative Measurement (Code Design)

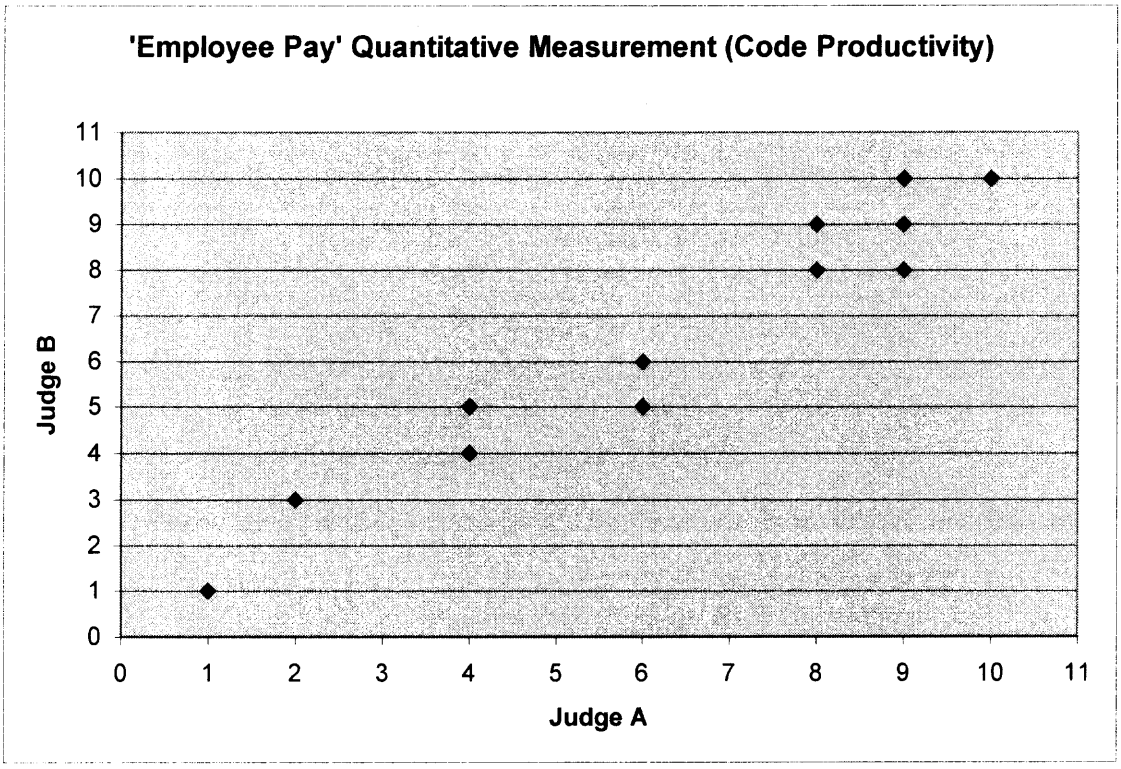


Figure K.7 'Employee Pay' Quantitative Measurement (Code Productivity)

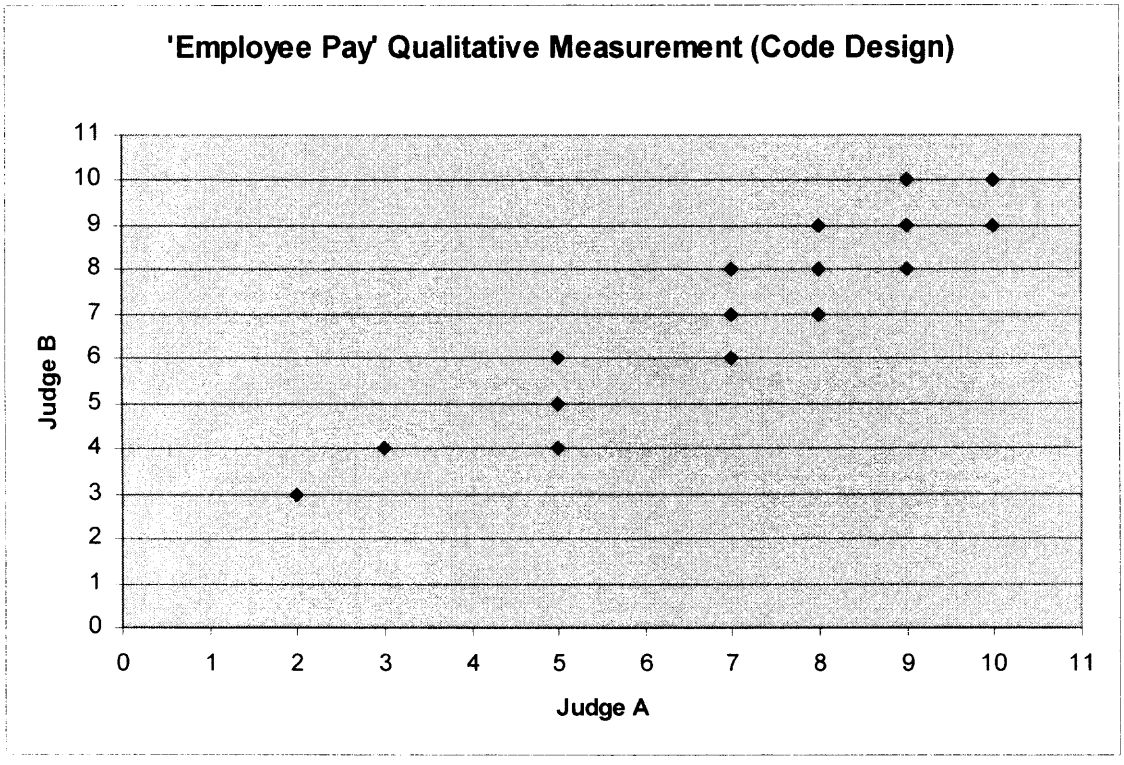


Figure K.8 'Employee Pay' Qualitative Measurement (Code Design)

APPENDIX L

PAIR PROGRAMMING SURVEY QUESTIONNAIRE

Preliminary Pair Programming Questionnaire

Experiment Title: Discovering and analyzing the success factors of maximization of pair programming productivity

Investigator: K. Steve Choi, PhD candidate, Information Systems Department, New Jersey Institute of Technology

1. Your Job Title: _____

2. Type of Industry: _____

3. Gender: ___ Male ___ Female

4. I have been programming for ___ year(s), and ____ month(s)

5. I have been practicing (or have practiced) pair programming for ____ year(s), and ____ month(s)

6. The list below is the possible variables that may impact the pair programming productivity. Blank fields with * mark are where you enter any variable(s) that you feel it belongs in the list, but you do not see it from the current list. After that, rank the following variables (including the ones that you have entered) where number 1 being the most impact variable. (The variable ranked #1 is most impact variable to the pair programming productivity)

Direction: 1) You may add any variable(s) that you feel it belongs in the list but do not see. 2) You are then asked to rank the variables, 1 being the most impact variable.

Rank Variables

_____ Gender (Male/Female),

_____ Programming skill level (Low/Med/High)

_____ Cognitive/Programming style (i.e. sequential/ object-oriented)

_____ Personality (i.e. Meyers-Briggs personality types)

_____ A discrete step by step pair protocol (present/absent)

_____ Familiarity (friends, same culture, etc)

_____ Fluency in English (Communication)

_____ *_____

_____ *_____

_____ *_____

_____ *_____

7. In the following text lines, please enter ANY comments that you may have to the variable list or to this experiment.

_____

_____

_____

_____

New added variables (# of votes)	Original variables (# of votes)
Technical knowledge (1)	Gender (0)
Business domain knowledge (3)	Programming skill level (17)
"Keyboard switch" (1)	Cognitive/Programming style (25)
Ability to work with others (9)	Personality (28)
Willingness to communicate (3)	A discrete step by step pair protocol (3)
Personal hygiene (4)	Familiarity (8)
Working environment (3)	Fluency in English (21)
Coding conventions/Platform (3)	
Time constraint (1)	
Availability of snacks (1)	
Open to new ideas (2)	
Experience (2)	
Common expectation (3)	
Tiredness (2)	
Desire to learn (1)	

APPENDIX M

MBTI DISTRIBUTION AMONG DIFFERENT ACADEMIC MAJORS

Pair Programming Experiment Subjects
MBTI Profile Distribution
(Majors: Computer Science, Information Systems, and MIS)

Sensing Types		Intuitive Types			
with Thinking	with Feeling	with Feeling	with Thinking		
ISTJ N = 27 **21.1 %**	ISFJ N = 7 5.5 %	INFJ N = 1 0.8 %	INTJ N = 5 3.9 %	Judging	Introverts
ISTP N = 4 3.1 %	ISFP N = 8 6.3 %	INFP N = 9 7.0 %	INTP N = 9 7.0 %	Perceptive	
ESTP N = 7 5.5 %	ESFP N = 7 5.5 %	ENFP N = 7 5.5 %	ENTP N = 4 3.1 %	Perceptive	Extroverts
ESTJ N = 19 **14.8 %**	ESFJ N = 7 5.5 %	ENFJ N = 3 2.3 %	ENTJ N = 4 3.1 %	Judging	

Table M.1 Distribution of Types Within Occupational and Academic Group

	ST (%)	SF (%)	NF (%)	NT (%)
Occupations				
Accountants	64	23	4	9
Bank employees	47	24	11	18
Sales, customer relations	11	81	8	0
Creative writers	12	0	65	23
Research scientists	0	0	23	77
Fields of Graduate Studies				
Theology (liberal)	3	15	57	25
Law	31	10	17	42
Fields of College Studies				
Finance and commerce	51	21	10	18
Nursing	15	44	34	7
Counseling	6	9	76	9
Science	12	5	26	57
Health-related professions	13	36	44	7
Education	13	42	39	6
Journalism	15	23	42	20
P.E. and health	32	34	24	10

*Source: Myers and Myers (1995)

Pair Programming Exp.	44	23	16	17

APPENDIX N

HISTOGRAMS AND Q-Q PLOTS FOR NORMAL DISTRIBUTION CHECK

Following figures are normality check histograms and Q-Q plots.

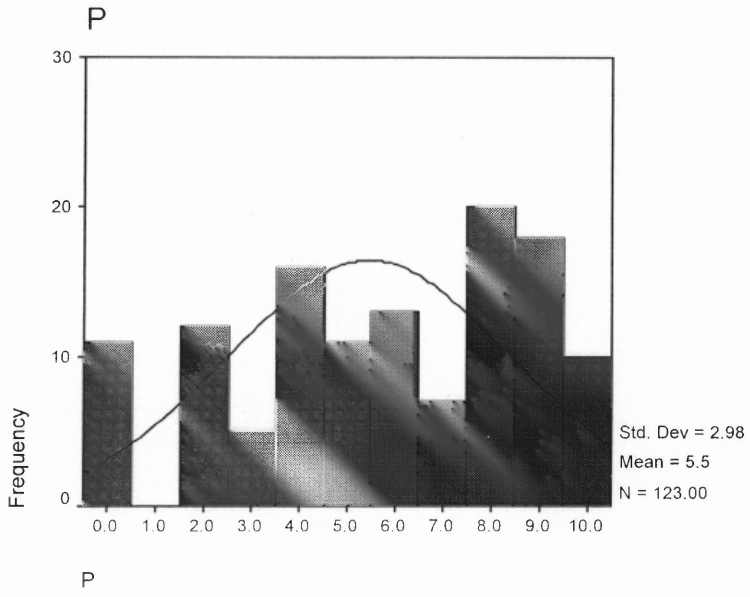


Figure N.1 Histogram of Dependent Variable P

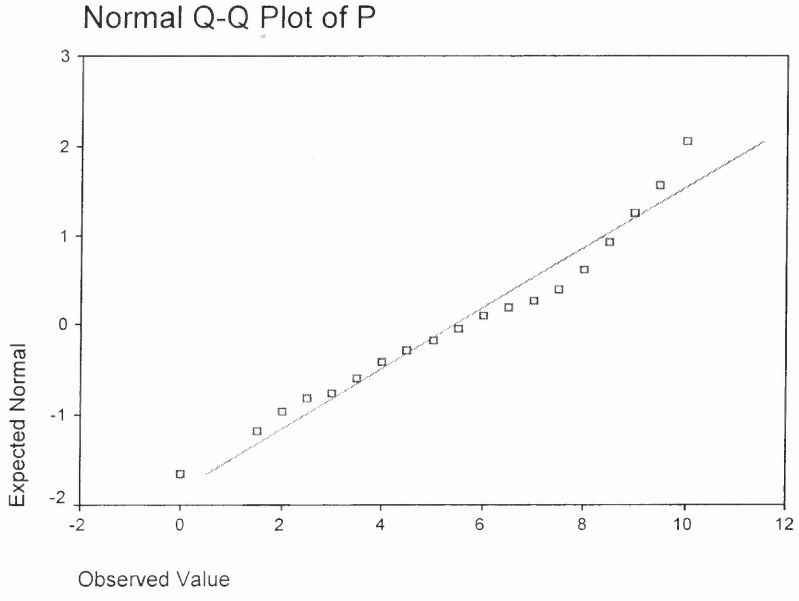


Figure N.2 Normal Q-Q Plot of P

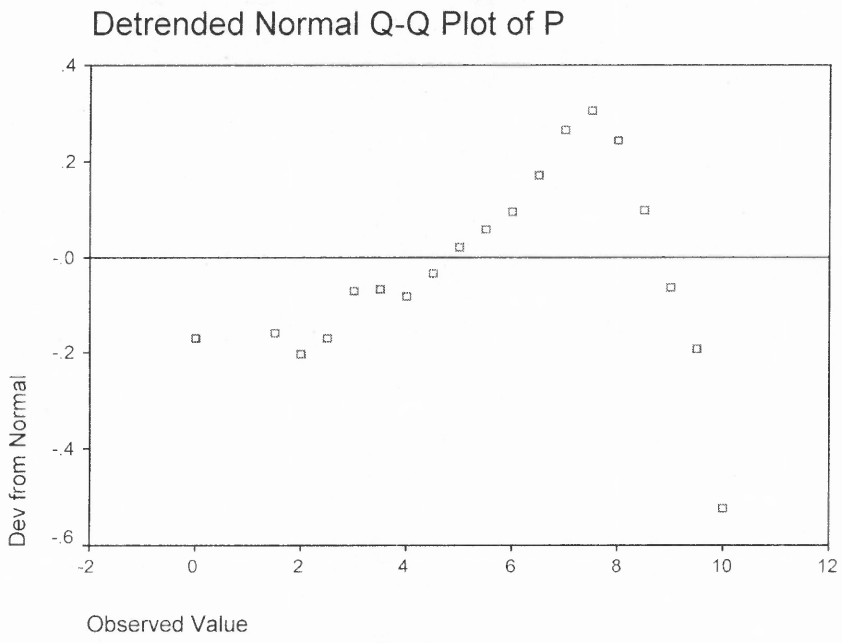


Figure N.3 Histogram of Dependent Variable code design dependent variable

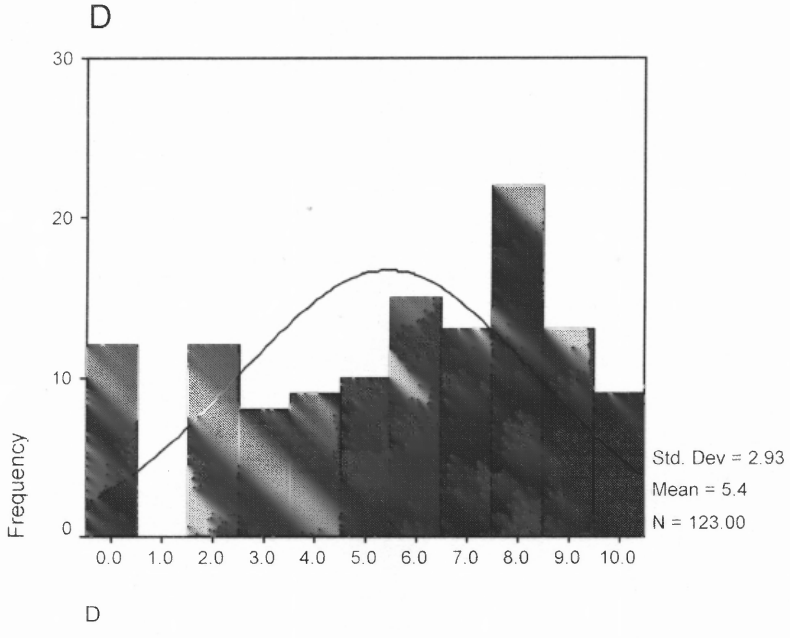


Figure N.4 Histogram of Dependent Variable D

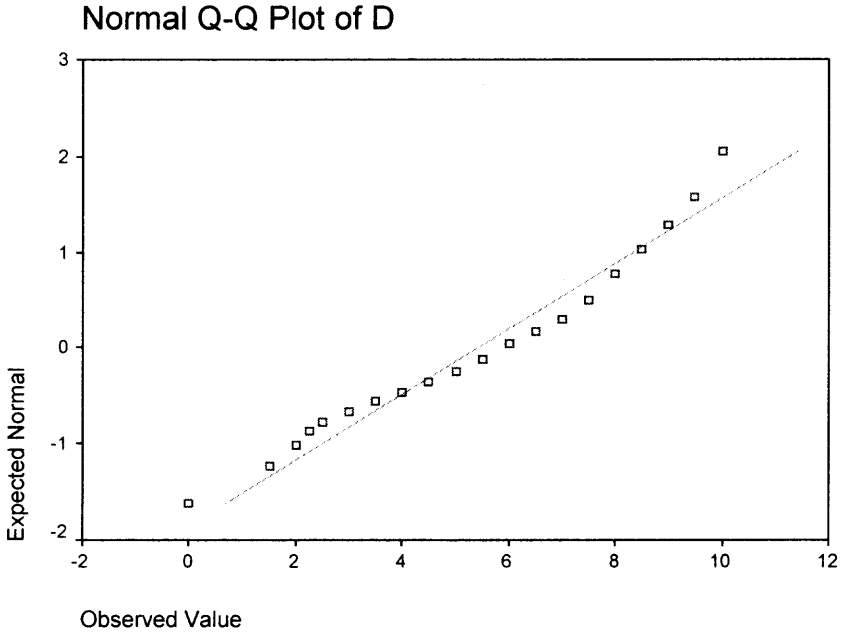


Figure N.5 Normal Q-Q Plot of D

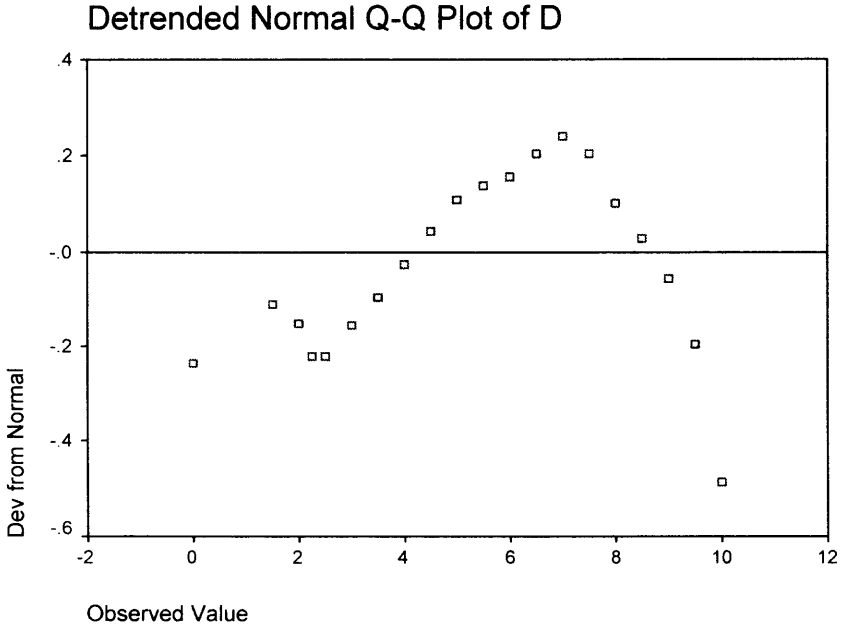


Figure N.6 Detrended Normal Q-Q Plot of D

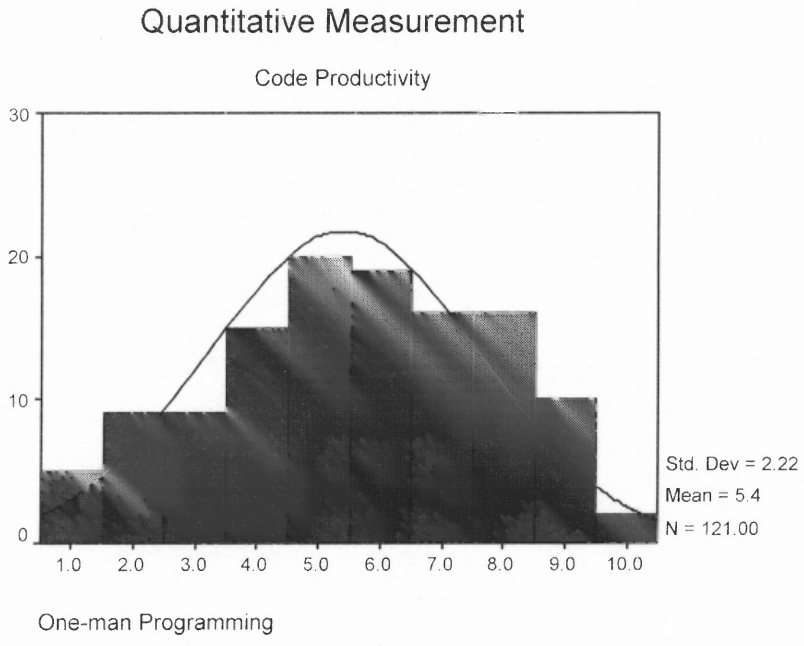


Figure N.7 Histogram of one-man programming code productivity dependent variable

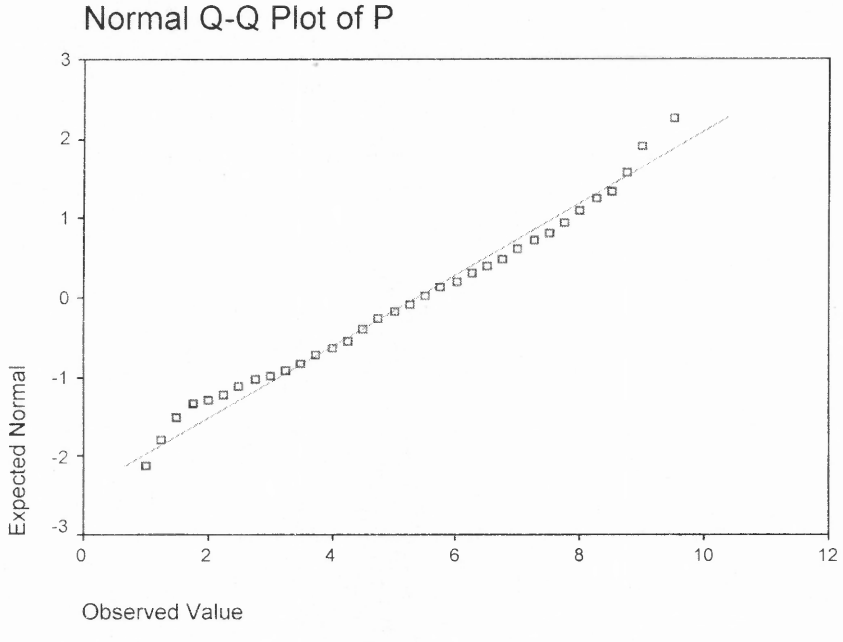


Figure N.8 Normal Q-Q Plot of one-man programming code productivity dependent variable

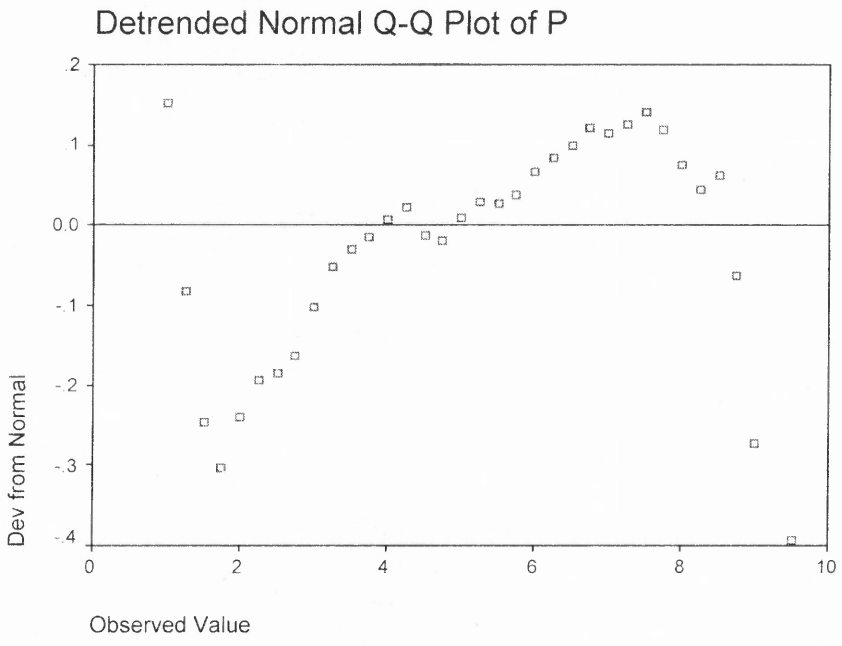


Figure N.9 Detrended Normal Q-Q Plot of one-man programming code productivity dependent variable

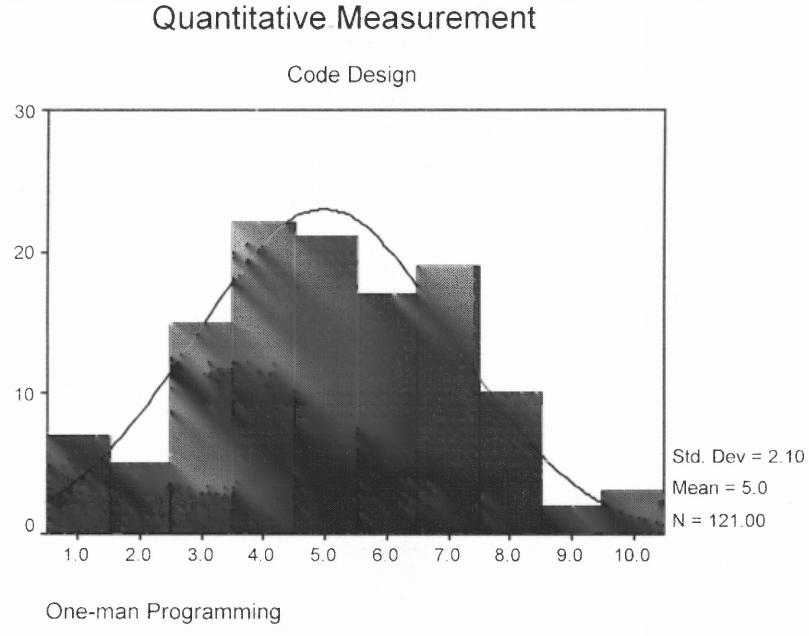


Figure N.10 Histogram of one-man programming code design dependent variable

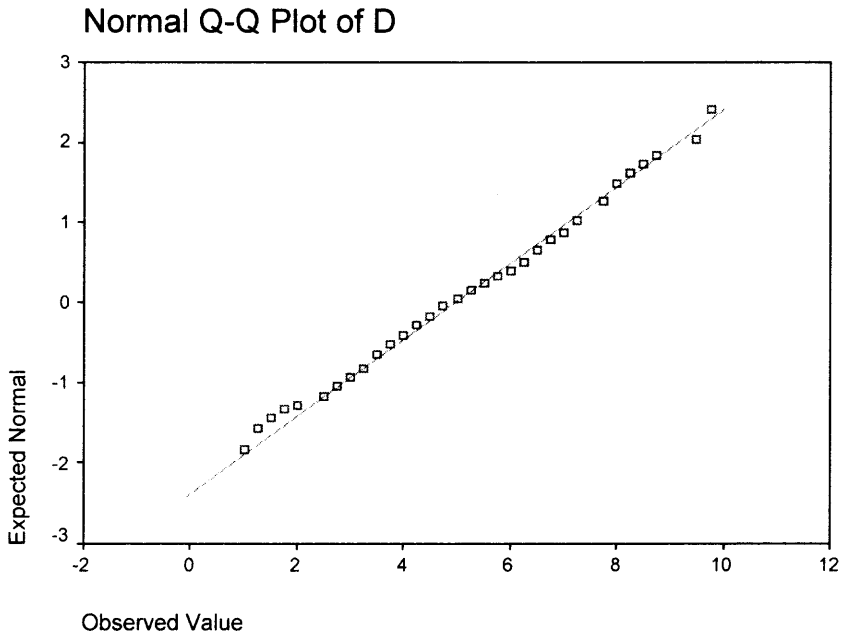


Figure N.11 Normal Q-Q Plot of one-man programming code design dependent variable

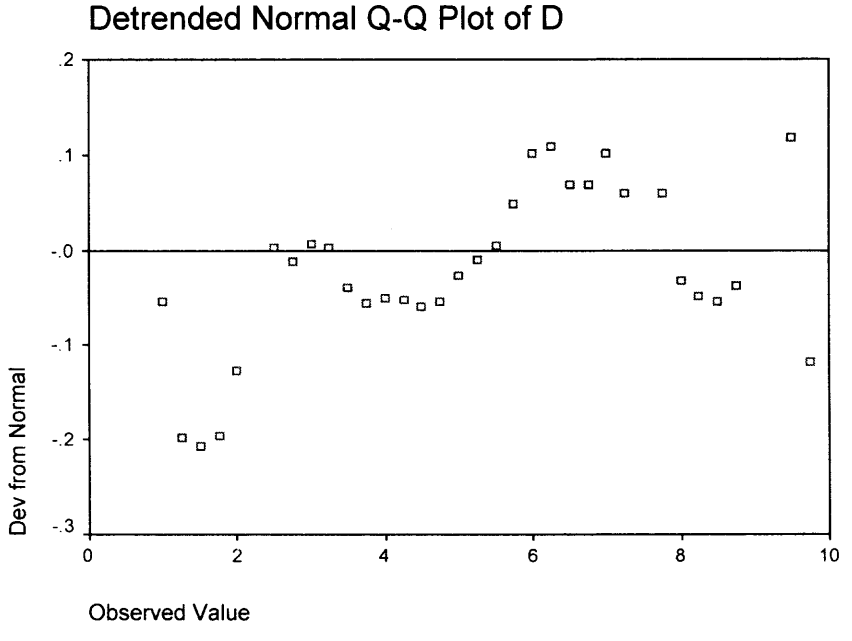


Figure N.12 Detrended Normal Q-Q Plot of one-man programming code design dependent variable

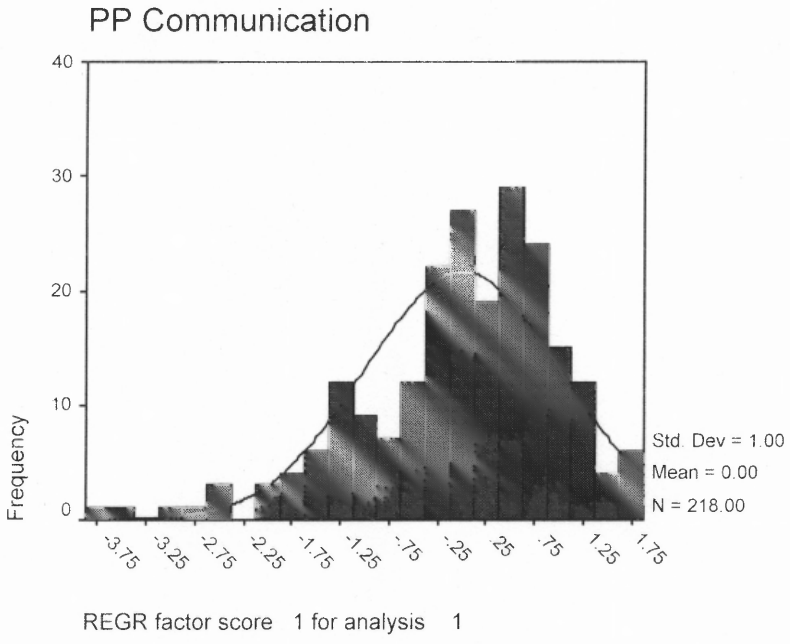


Figure N.13 Normal Distribution of PP Communication

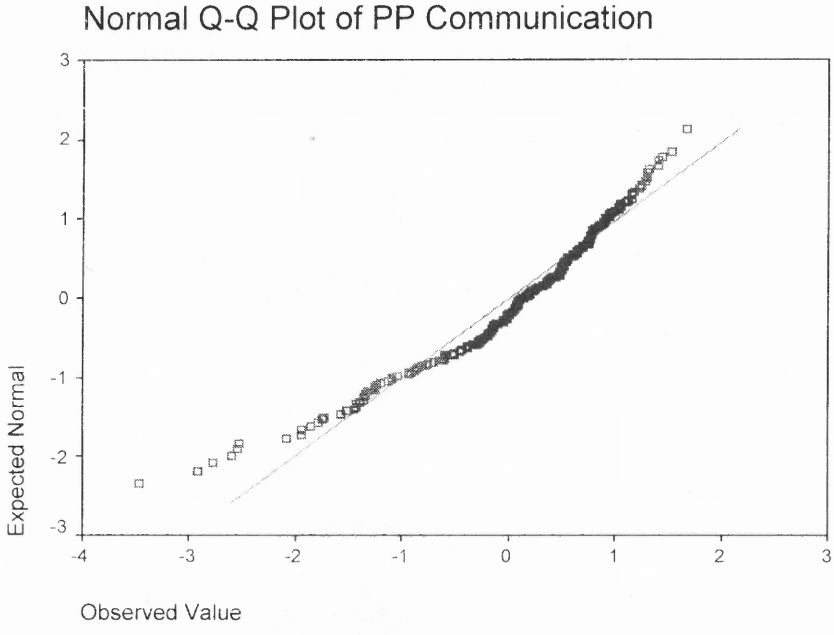


Figure N.14 Normal Q-Q Plot of PP Communication

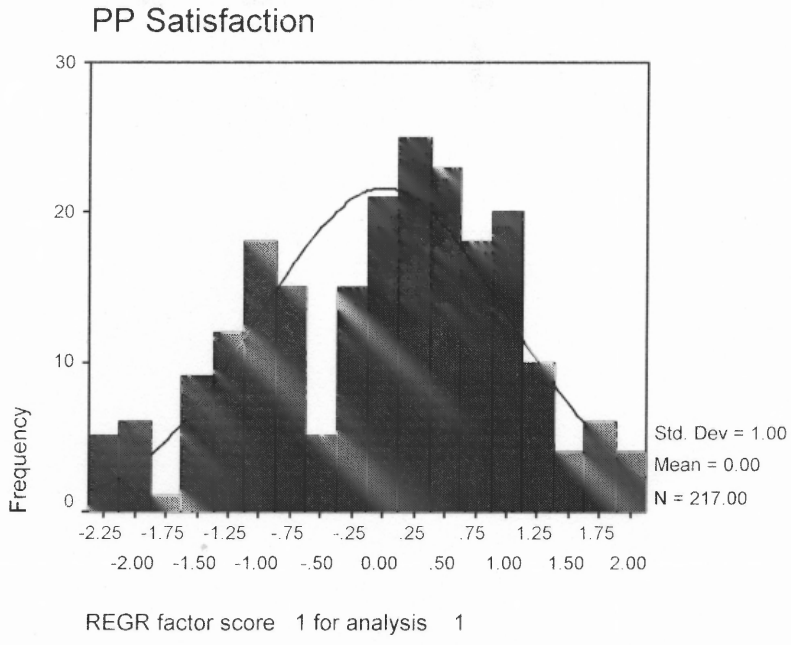


Figure N.15 Normal Distribution of PP Satisfaction

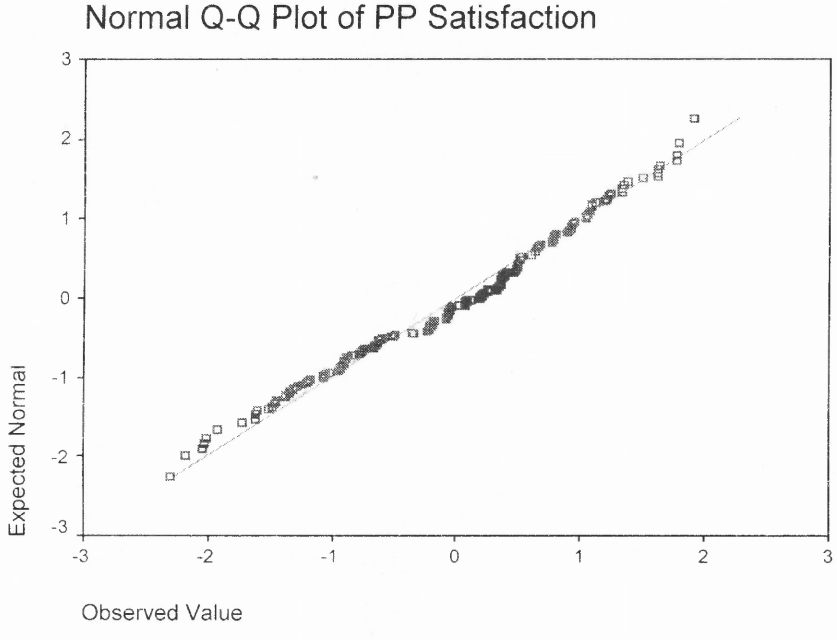


Figure N.16 Normal Q-Q Plot of PP Satisfaction

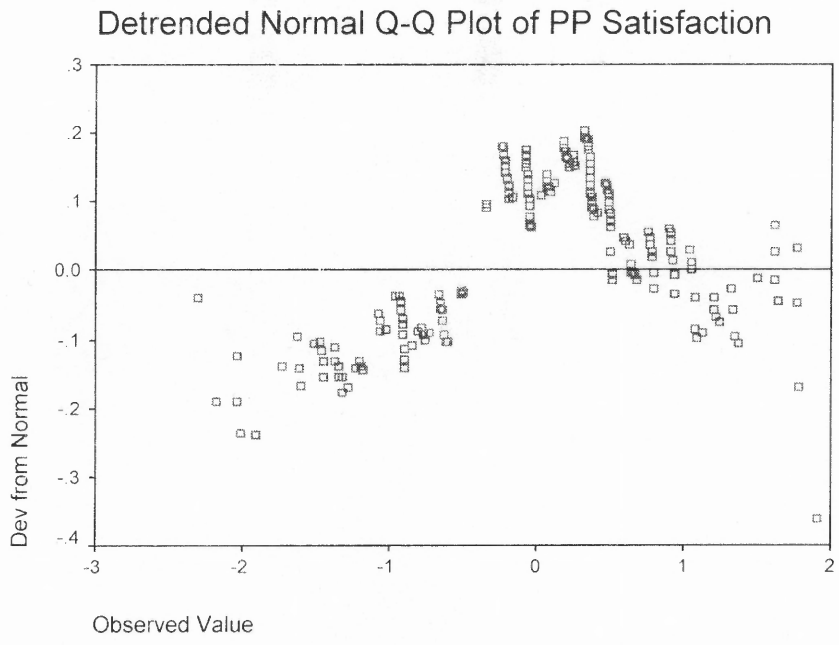


Figure N.17 Detrended Normal Q-Q Plot of PP Satisfaction

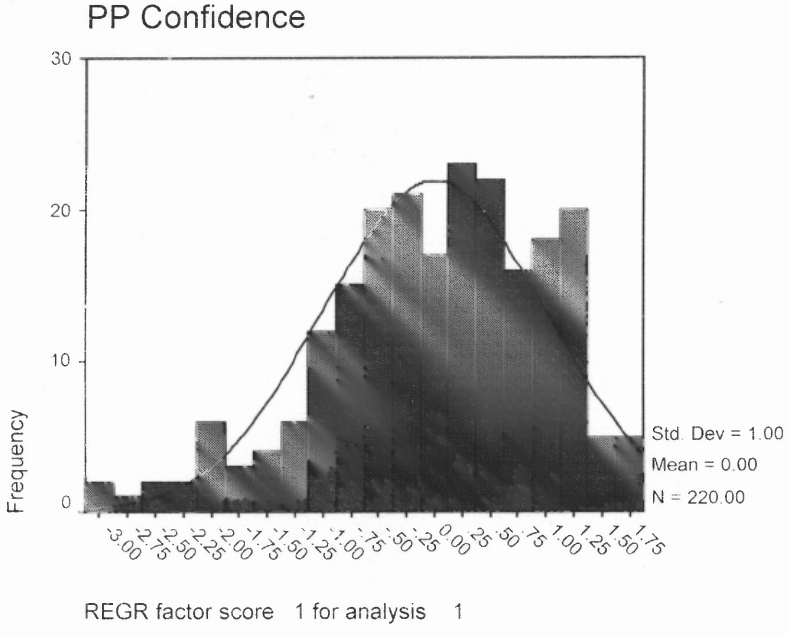


Figure N.18 Normal Distribution of PP Confidence

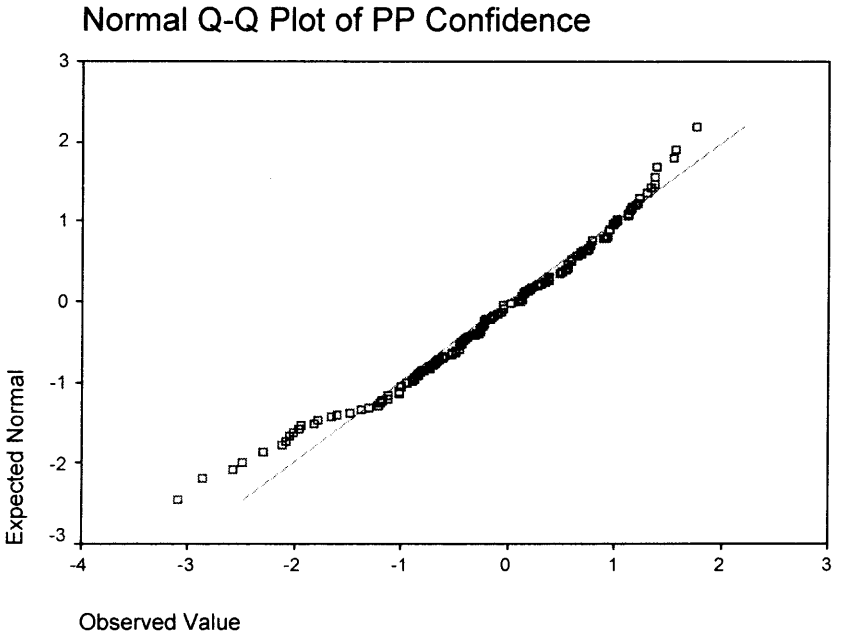


Figure N.19 Normal Q-Q Plot of PP Confidence

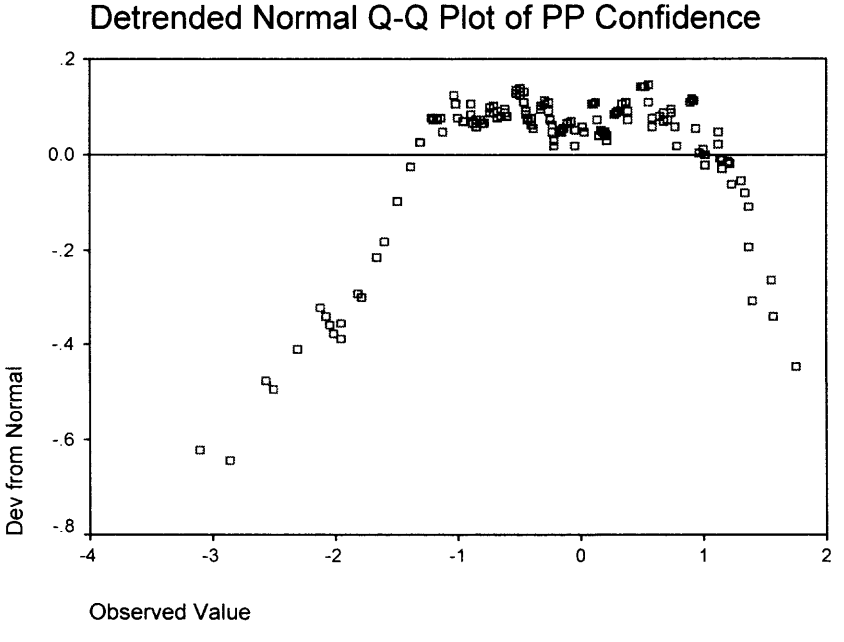


Figure N.20 Detrended Normal Q-Q Plot of PP Confidence

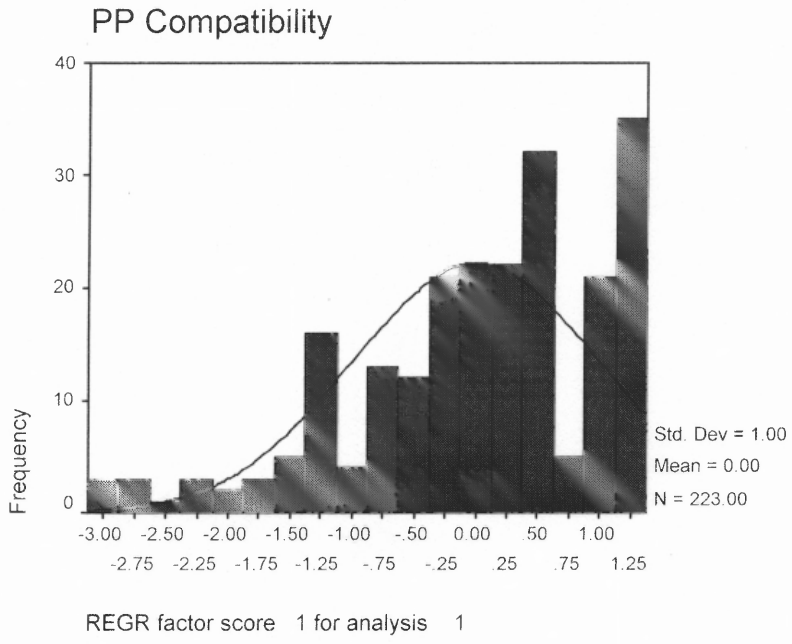


Figure N.21 Normal Distribution of PP Compatibility

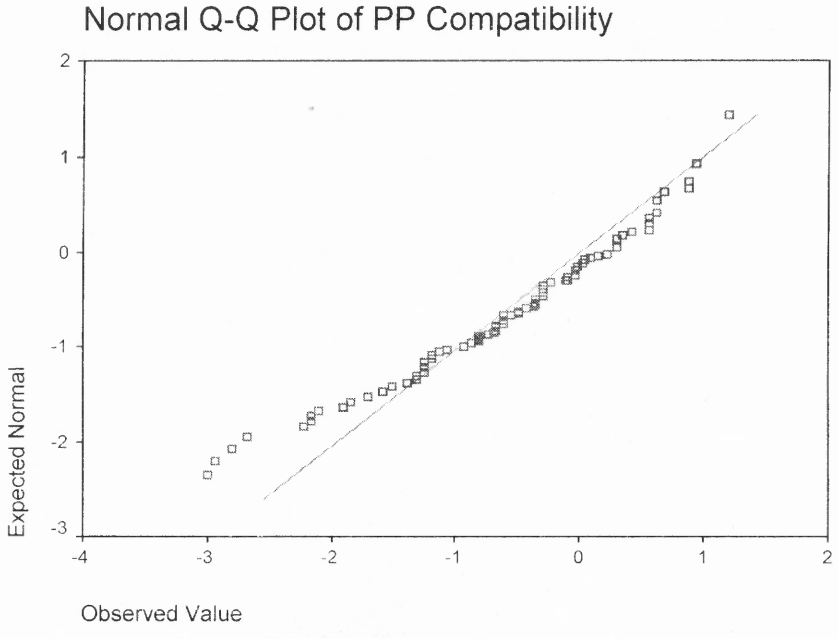


Figure N.22 Normal Q-Q Plot of PP Compatibility

APPENDIX O

EXPERIMENT ENVIRONMENT SPECIFICATION

Following is the description of experiment location, hardware and software.

Table 0.1 Experiment Environment Specification

	Categories	Specifications	Comments
Location	Learning Systems Laboratory GITC Bldg.	11 Personal Computers on a U-shaped table	Restricted area and only accessible by authorized personnel
Hardware	Personal Computer	Manufacturer: Dell CPU: Pentium III 1.6 GHz RAM: 512 MB Display: 17" Dell Monitor	None
Software*	Operating System	Microsoft Windows 2000 Professional	None
	Programming Languages	JAVA(TM) 2 Software Development Kit (J2SDK), Standard Edition, Version 1.4.2_03	Downloaded from Java home page http://java.sun.com/j2se/1.4.2/download.html
		C++ Microsoft Visual Studio 6	Installed by University Computing Systems Services

*No special editor (auxiliary) software application is used.

GLOSSARY AND INDEX

Alike pair – A pair where two persons are alike in both their kind of perception and judgment, the two inner preferences (i.e. ENTP + INTJ). Pg 130.

[alike] – see alike pair.

Auxiliary type – Used concomitantly with dominant type, auxiliary type is a person's secondary preference. It is readily and frequently used as needed by the person. Again, the four preferences are sensing (S), intuition (N), thinking (T), and feeling (F). The person uses the dominant and auxiliary types freely and interchangeably according to a changing situation. Pg 125.

code design – An area that involves appropriate selection between structured, modular, and object-oriented programming, and attaining a high level of code efficiency and code readability. Pg 131.

code productivity – An area that quantitatively measures how much coding a subject did in completing a programming task which includes generating a code output. Pg 131.

Communication Skill - intentionally repeatable, goal-directed behaviors and behavior sequences in conveying one's messages. Pg 127.

Diverse pair – A pair where two persons are alike in their kind of perception or judgment, the two inner preferences, but not both (i.e. ENTP + INFJ). Pg 130.

[divrs] – see diverse pair.

Dominant type - According to MBTI theory, every person has one of four possible dominant preferences which he or she uses the most and is best at . The four preference areas are sensing (S), intuition (N), thinking (T), and feeling (F). Pg 125.

FF – Female-Female pair. Pg 131.

[FF] – see FF.

HH – A pair of individuals who both possess a high level of communication skill. Pg 130.

[HH] – see HH.

HL – A pair of individuals where one possesses a high level of communication skill and the other possesses a low level of communication skill. Pg 130.

[HL] – see HL.

LL – A pair of individuals who both possess low levels of communication skill. Pg 130.

[LL] – see LL.

MBTI – Myers-Briggs Type Indicator (MBTI). Created by Isabel Briggs Myers and Katharine Briggs, it is the most widely used personality inventory instrument for measuring a person's preferences. It uses four basic scales with opposite poles. Pg 122.

MM – Male-Male pair. Pg 131.

[MM] – Male-Male pair.

MF – Male-Female pair. Pg 131.

[MF] – Male-Female pair.

Opposite pair – A pair where two persons are opposite in both their kind of perception and judgment, the two inner preferences (i.e. E**NTP** + E**SFJ**). Pg 130.

[opp] – see opposite pair.

[paircomm] – The group of pairs where each pair is paired according to the person's communication skill level, There are a total of three groups: High-High [HH], High-Low [HL], and Low-Low [LL]. Pg 167.

[pairgender] – The group of pairs where each pair is paired according to the person's gender. There are a total of three groups: Male-Male [MM], Male-Female [MF], and Female-Female [FF]. Pg 168.

[pairtype] – The group of pairs where two persons are paired according to the person's dominant and auxiliary preferences of MBTI type. There are a total of three groups: [opp], [divrs], and [alike]. Pg 163, 164.

REFERENCES

1. Agile software development manifesto website: http://agilemanifesto.org/, Date retrieved Nov-26-2003.

2 Abrahamsson, P., Warsta, J., Siponen, M. T., and Ronkainen, J., (2003) "New directions on agile methods: a comparative analysis", Proceedings of the 25th international conference on Software engineering, Portland, Oregon, 2003, pp. 244-254.

3. Anderson, A., Bettie, R., and Beck, K., (1998) "Chrysler goes to Extreme", Distributed computing, Oct 1998, pp. 24-28.

4 Anderson, C., and Dorfman, M., (1991) *Aerospace software engineering: a collection of concepts*, Anderson, C. and Dorfman, M., Eds. Washington, DC: American Institute of Aeronautics and Astronautics, 1991.

5. Aoyama, M., (1998) "Agile Software Engineering Environment over the Internet", ICSE 98 Workshop on Software Engineering over the Internet, April 25, 1998, Calgary, Canada. http://sern.cpsc.ucalgary.ca/~maurer/ICSE98WS/Submissions/Aoyama/ICSE98-Internet-Mikio.ps., Date retrieved Nov-25-2003.

6. Balk, D. L., and Kedia, A., (2000) "PPT: a COTS integration case study", Proceedings of the 22nd International conference on software engineering, Limerick, Ireland, 2000, pp. 42-49.

7. Baker, E. R., (2001) "Which Way, SQA?", IEEE Software, January/Feburary 2001, pp. 16-18.

8. Barnes, C., (2001) More programmers going "Extreme," CNET News.com, April 3, 2001, http://news.com.com/2100-1040-255167.html?legacy=cnet Date retrieved Nov-25-2003.

9 Basili, V.R. and Boehm, B., (2001) "COTS-based systems top 10 list", Computer, 34(5), May 2001, pp. 91 – 95.

10. Baskerville, R., Levine, L., Pries-Heje, J., and Slaughter, S., (2001) "How Internet software companies negotiate quality", Computer, 34(5), May 2001, pp. 51 – 57.

11. Baskerville, R., and Pries-Heje, J., (2001a) "Racing the E-Bomb: How the Internet Is Redefining Information Systems Development Methodology", Realigning Research and Practice in IS Development: The Social and Organisational Perspective, pp. 49-68, 2001.

12. Baskerville, R., and Heje, J. P., (2001b) "EMethodology: Towards a systems development methodology for e-business and e-commerce applications", Elliot, S, Anderen, K. V., Swatman, P., and Reich, S., (Eds.), Developing a Dynamic, Integrative, Mult-Disciplinary Research Agenda in E- Commerce/E-Business. Newcastle, Austalia: BICE Press.2001. pp. 145-159.

13 Baskerville, R., Travis, J., and Truex, D. P., (1992) "Systems Without Method: The Impact of New Technologies on Information Systems Development Projects", Proceedings of the International Federation for Information Processing (IFIP), Working Conference on The Impact of Computer Supported Technologies in Information Systems Development, June 14-17, 1992. pp. 241-269.

14 Baskerville, R., Ramesh, B., Levine, L., Pries-Heje, J., and Slaughter, S., (2003) "Is internet-speed software development different?" Software, IEEE, 20(6), Nov/Dec. 2003, pp. 70 – 77.

15. Battin, R.D., Crocker, R., Kreidler, J., and Subramanian, K., (2001) "Leveraging resources in global software development", IEEE Software, 18(2), March/April 2001, pp. 70 – 77.

16. Bayne, R., (1995) The Myers-Briggs type indicator: a critical review and practical guide. 1st ed. Singular Pub., 1995.

17. Beck, K., (2000) Extreme Programming eXplained: embrace change, Reading, MA: Addison-Wesley, 2000.

18. Beck, K., and Fowler, M., (2001) Planning extreme programming, Boston, MA: Addison-Wesley, 2001.

19. Behforooz, A., and Hudson, F. J., (1996) Software engineering fundamentals, New York, Oxford University Press, 1996.

20 Boehm, B., (2002a) "Software engineering is a value-based contact sport", Software, IEEE, Vol. 19, Issue 5, Sept.-Oct. 2002, pp. 95 – 96.

21 Boehm, B., (2002b) "Get ready for agile methods, with care", Computer, 35(1), Jan. 2002, pp. 64 – 69.

22. Boehm, B., (1988) "A Spiral Model of Software Development and Enhancement", Computer, 21(5), May 1988, pp. 61 – 72.

23. Boehm, B., (1976) "Software Engineering," IEEE Transactions on Computers, C-25(12), December 1976, pp. 1226 – 1241.

24 Boehm, B., (2000) "Unifying software engineering and systems engineering", Computer, 33(3), March 2000, pp. 114 –116.

25 Boehm, B. and Basili, V.R., (2001) "Top 10 list software development", Computer, 34(1), Jan. 2001, pp. 135 – 137.

26. Boehm, B., and Belz, F., (1990) "Experiences with the spiral model as a process model generator", Proceedings of the 5th international software process workshop on Experience with software process models, 1990, pp. 43 – 45, http://www.acm.org/pubs/articles/proceedings/soft/317498/p43-boehm/p43-boehm.pdf. Date retrieved, Nov-25-2003.

27 Boehm, B., Port, D., Ye Yang, Bhuta, J., and Abts, C., (2003) "Composable process elements for developing COTS-based applications", Empirical Software Engineering, 2003. ISESE 2003. Proceedings. 2003 International Symposium on, 30 Sept. - 1 Oct. 2003, pp. 8 – 17.

28 Boehm, B. and Turner, R., (2003) "Observations on balancing discipline and agility", Agile Development Conference, 2003. ADC 2003. Proceedings of the, 25-28 Jun 2003, pp. 32 – 39.

29. Borstler, J.and Janning, T., (1992) "Traceability between requirements and design: a transformational approach", Computer Software and Applications Conference, 1992. COMPSAC '92. Proceedings., Sixteenth Annual International , 1992, pp. 362 – 368.

30 Boudreau, M., Gefen, D., and Straub, D. W., (2001) "Validation in Information Systems Research: A State-of-the-Art Assessment", Management Information Systems Quarterly, Mar 2001, 25(1), pp. 1-16.

31. Brownsword, L., and Place, P., (1999) Lessons learned applying Commercial Off-the-shelf products, 1999, http://www.sei.cmu.edu/pub/documents/99.reports/pdf/99tn015.pdf. Date retrieved, Nov-25-2003.

32 Brooks, R., (1999) "Towards a theory of the cognitive processes in computer programming", International Journal of Human-Computer Studies, 51(2), August 1999, pp. 197-211.

33. Bubenko, J.A., Jr., (1995) "Challenges in requirements engineering, Requirements Engineering", Proceedings of the Second IEEE International Symposium, 27-29 March 1995, pp. 160 –162.

34 Bubenko, J.A., Jr., and Wangler, B., (1993) "Objectives driven capture of business rules and of information systems requirements, Systems, Man and Cybernetics", 'Systems Engineering in the Service of Humans', Conference Proceedings, International Conference on, 17-20 Oct. 1993, Vol.1. pp. 670 – 677.

35. Budlong, F.C.and Stanko, J.J., (1993) "Using software document evaluations to enhance software supportability", Digital Avionics Systems Conference, 1993. 12th DASC., AIAA/IEEE , 1993, pp. 433 – 438.

36. Carmel, E., (1997) "Thirteen assertions for globally dispersed software development research", System Sciences, Proceedings of the Thirtieth Hawaii International Conference on, Vol. 3, 1997, pp. 445 – 452.

37 Capretz, L. F., (2003) Personality Types in Software Engineering, International Journal of Human-Computer Studies, 58(2), Feb.2003, pp.207-214.

38. Carmel, E. and Agarwal, R., (2001) "Tactical approaches for alleviating distance in global software development", IEEE Software, 18(2), March/April 2001, pp. 22 – 29.

39. CCP website 1, http://www1.cpp.com/press/brandstrategy.asp. Date retrieved, Nov-25-2003.

40. Choi, K. S., and Deek, F.P., (2002) "Extreme Programming, Too Extreme?", Proceedings of the International Conference on Software Engineering Research and Practice 2002 (SERP'02), Las Vegas, USA, 2002.

41. Cockburn, A., (2001) Agile Software Development, Addison-Wesley, 2001.

42 Cockburn, A., (2000) "Selecting a Project's Methodology", IEEE Software, 17(4), pp. 64-71, July 2000.

43 Cockburn, A. and Highsmith, J., (2001) "Agile software development, the people factor", Computer, 34(11), Nov. 2001, pp. 131 – 133.

44 Cohen, R. J., and Swerdlik, M. E., (2002) Psychological testing and assessment: an introduction to tests and measurement, 5th ed., Boston: McGraw-Hill, 2002.

45. Cohn, T.M., and Paul, R.C., (2001) "A Comparison of Requirements Engineering in Extreme Programming (XP) and Conventional Software Development Methodologies", Seventh Americas Conference on Information Systems, 2001, pp. 1327- 1331.

46 Constantine, L. L., (1995) Constantine On Peopleware, Yourdon Press, 1995.

47 Corritore, C. L., and Wiedenbeck, S., (1999) "Mental representations of expert procedural and object-oriented programmers in a software maintenance task", International Journal of Human-Computer Studies, 50(1), January 1999, pp. 61-83.

48. Cosgrove, J., (2001) "Software engineering and the law", IEEE Software, 18(3), May/June 2001, pp. 14 –16.

49. Crystal website: http://crystalmethodologies.org. Date retrieved, Nov-25-2003.

50. Cusumano, M., and Yoffice, D. (1998) Competing on Internet Time: Lessons from Netscape and Its Battle with Microsoft. New York, NY: Free Press, 1998.

51 Cusumano, M. A., and Yoffie, D. B., (1999) "Software Development on Internet Time", Computer, 32(10), October 1999, pp. 60 – 69.

52. Davis, A.M., Bersoff, E.H., Comer, E.R, (1988) "A strategy for comparing alternative software development life cycle models", Software Engineering, IEEE Transactions on, 14(10), Oct. 1988, pp. 1453 - 1461.

53 Deek, F.P., and McHugh, J., (2003) "A Case Study in an Integrated Development and Problem Solving Environment", to appear in Journal of Interactive Learning Research, 2003.

54 Deek, F.P., Elliot, N., Coppola, N., and O'Daniel, N., (2000) "Cognitive Characteristics of Web-Developers: Creativity, Meaning Construction, and Problem Solving", WebNet Journal: Internet Technologies, Applications and Issues, 2(2), pp. 36-50, April-June 2000.

55 Deek, F. P., and McHugh, J., (2000) "Problem Solving and the Development of Critical Thinking Skills", Journal of Computer Science Education - ISTE SIGCS, 14(1/2), pp. 6-12, April 2000.

56 DeFranco-Tommarello, J., and Deek, F. P., (2003) "A Review and Analysis of Collaborative Problem Solving and Groupware for Software Development", to appear in Journal of Information Systems Management, 2003.

57 DeFranco-Tommarello, J., Deek, F. P., Hiltz, S. R., Keenan, J., and Perez, C., (2003) "Collaborative Software Development: Experimental Results", to appear in the Proceedings of the Hawaii International Conference on System Sciences (HICSS 36), Big Island, Hawaii, USA, 2003.

58. DeMarco, T., and Lister, T., (1999) Peopleware: Productive Projects and Teams. 2nd ed. New York, NY: Dorset House Publishing, 1999.

59. Dooley, K., (1996) Complex Adaptive Systems: A Nominal Definition, 1996 website http://www.eas.asu.edu/~kdooley/casopdef.html. Date retrieved, Nov-25-2003.

60. Dorfman, M., (1997) "Requirements Engineering", Software Requirements Engineering 2nd ed., Thayer, R.H., and Dorfman, M., (Eds.), Los Alamitos, CA: IEEE Computer Society Press, 1997, pp. 11-12.

61. DSDM website http://na.dsdm.org/na/default.asp. Date retrieved, Nov-25-2003.

62. Eljabiri, O., and Deek, F., P., (2001) "Toward a Comprehensive Framework for Software Process Modeling Evolution", Proceedings of the ACS/IEEE International Conference on Computer Systems and Applications (AICCSA), Beirut, Lebanon, 2001.

63. El Sawy, O., (2001) Redesigning Enterprise Processes for e-Business, New York, McGraw-Hill, 2001.

64. Ebert, C. and De Neve, P., (2001) "Surviving global software development", IEEE Software, Vol 18, Issue 2, March/April 2001, pp. 62 – 69.

65 Engebretson, A.and Wiedenbeck, S., (2002) "Novice comprehension of programs using task-specific and non-task-specific constructs", Human Centric Computing Languages and Environments, 2002. Proceedings. IEEE 2002 Symposia on, 3-6 Sept. 2002, pp. 11 – 18.

66 Erdogmus, H., Boehm, B.W., Harrison, W., Reifer, D.J., and Sullivan, K.J., (2002) "Software engineering economics: background, current practices, and future directions", Software Engineering, 2002. ICSE 2002. Proceedings of the 24th International Conference on, 19-25 May 2002, pp. 683 – 684.

67 Field, A., (2003) Discovering Statistics: using SPSS for windows, Sage Publications, 2003.

68 Fix, V., Wiedenbeck, S., and Scholtz, J., (1993) "Mental Representations of Programs by Novices and Experts", INTERCHI'93, Proceedings, April 24-29 1993, pp. 74-79.

69. Floyd, C., (19884) "A Systematic look at prototyping", Approaches to Prototyping, Hedidelberg, Germany: Springer-Verlag, 1984, pp. 105-122.

70. Frosberg, K., and Mooz, H., (1996) System Engineering Overview, Center for Systems Management, Cupertino CA.1996.

71. Fowler, M., (2001) "Is Design Dead?", Extreme Programming examined, Succi, G., and Marchesi, M., (Eds.), Boston, MA: Addison-Wesley, 2001.

72. Fox, G., Lantner, K., and Marcom, S., (1997a) "A Software Development Process for COTS-Based Information System Infrastructure: Part 1", IEEE/SEI-sponsored Fifth International Symposium on Assessment of Software Tools and Technologies, Pittsburgh, Pa. held June 3-5,1997, pp. 133-142.

73. Fox, G., Marcom, S., and Lantner, K. W., (1997b) "A Software Development Process for COTS-Based Information System Infrastructure Part II, Lessons Learned", IEEE/SEI-sponsored Fifth International Symposium on Assessment of Software Tools and Technologies, Pittsburgh, Pa., June 3-5,1997, pp. 8 - 10.

74. Glass, R.L., (2001) "Extreme programming: the good, the bad, and the bottom line", IEEE Software, 18(6), Nov/Dec 2001, pp. 112 – 111.

75. Goguen, J.A., (1993) "Social issues in requirements engineering, Requirements Engineering", Proceedings of IEEE International Symposium, Jan. 1993, 4th-6th, pp. 194 – 195.

76. Goldman, S, L., Nagle, R., and Preiss, K., (1995) Agile competitors and virtual organizations: strategies for enriching the customer, Wiley & Sons, New York, NY, 1995.

77. Gomaa, H., (1997) "The impact of prototyping on software system engineering", Software requirements engineering 2nd edition, Thayer, R.H., and Dorfman, M., (Eds.), Los Alamitos, CA: IEEE computer society press, 1997, pp. 479 - 488.

78. Grenning, J. (2001) "Launching extreme programming at a process-intensive company", IEEE Software, 18(6), Nov.-Dec. 2001, pp. 27 – 33.

79. Han, J., (1994) "Software documents, their relationships and properties", Software Engineering Conference, Proceedings., 1994 First Asia-Pacific, 1994, pp. 102 – 111.

80. Haungs, J., (2001) "Pair programming on the C3 project", Computer, Feb 2001, pp. 118 – 119.

81. Highsmith, J. A., (2000) Adaptive software development: a collaborative approach to managing complex systems, Orr. K., foreword, New York: Dorset House, 2000.

82 Highsmith, J. and Cockburn, A., (2001) "Agile software development: the business of innovation", Computer, 34(9), Sept. 2001, pp. 120 – 127.

83. Hirai, C., Saeki, N., and Nakano, T., (1998) "A proposal of an Internet-based software development process model for COTS-Based systems development", International Conference of Software Engineering -1998 Workshop on Software Engineering over the Internet, Kyoto, Japan, April 19-25, 1998, http://sern.cpsc.ucalgary.ca/~maurer/ICSE98WS/Submissions/Hirai/hirai.html. Date retrieved, Nov-25-2003.

84. Holland, J. H., (1995) Hidden order: how adaptation builds complexity, Reading, Mass.: Addison-Wesley, 1995.

85. IEEE Std 610.12-1990, IEEE standard glossary of software engineering terminology, 10 Dec. 1990.

86 Jones, Capers. (1994) Assessment and control of software risks, Englewood Cliffs, N.J.: Yourdon Press, 1994.

87. Karlsson, E. .A.,. Andersson, L. G., Leion, P., (2000) "Daily build and feature development in large distributed projects", Software Engineering, 2000. Proceedings of the 2000 International Conference on, 2000, pp. 649 –658.

88. Keefer, G., (2002) Extreme programming considered harmful for reliable software development, 2002, http://www.avoca-vsm.com/Dateien-Download/ExtremeProgramming.pdf. Date retrieved, Nov-25-2003.

89. Keen, P., (1981) "Information systems and organizational change", Comm. of ACM, 24(1), 1981, pp. 24-33.

90. Keirsey, D., (1998) Please understand me II: temperament, character, intelligence., 1st ed. Del Mar, CA: Prometheus Nemesis, 1998.

91 Kellner, M. I., and Hansen, G. A., (1998) "Software Process Modeling", Technical Report, Software Engineering Institute, Carnegie Mellon University, May 1988. http://www.sei.cmu.edu/pub/documents/88.reports/pdf/tr09.88.pdf. Date retrieved, Nov-25-2003.

92. Kircher, M., Jain, P., Corsaro, A., and D. Levine, (2001) "Distributed Extreme Programming," XP2001 - eXtreme Programming and Flexible Processes in Software Engineering, Villasimius, Sardinia, Italy, May 21-23, 2001.

93. Kivi, J., Haydon, D., Hayes, J., Schneider, R., and Succi, G. (2000) "Extreme programming: a university team design experience", Electrical and Computer Engineering, 2000 Canadian Conference on, Vol. 2, 2000, pp. 816 – 820.

94. Kraut, R.E., and Streeter, L. A., (1995) "Coordination in Software Development" in Comm. Of ACM, 38(3), March 1995 pp. 69 - 81.

95 Kwon, K., Im, I., and Van de Walle, B., (2002) "Are you thinking what I am thinking? A comparison of decision makers' cognitive maps by means of a new similarity measure", System Sciences, 2002. HICSS. Proceedings of the 35th Annual Hawaii International Conference on, 7-10 Jan. 2002, pp. 1328 – 1337.

96. Lean Programming website 1 http://www.aanpo.org/articles/articles/LeanProgramming.htm. Date retrieved, Nov-25-2003.

97. Lean Programming website 2 http://c2.com/cgi/wiki?LeanProgramming. Date retrieved, Nov-25-2003.

98. Lichter, H., M. Schneider-Hufschmidt, and H. Zullighoven, (1994) "Prototyping in industrial software projects – Bridging the gap between theory and practice", IEEE Transaction on Software Engineering, 20(11), 1994, pp. 825-832.

99 Linger, R.C., and Trammell, C.J., Cleanroom Software Engineering Reference Model Version 1.0, 1996. http://www.sei.cmu.edu/pub/documatlents/96.reports/pdf/tr022.96.pdf. Date retrieved, Nov-25-2003.

100. Lutsky, P., (1995) "Automating testing by reverse engineering of software documentation", Reverse Engineering, 1995., Proceedings of 2nd Working Conference on , 1995, pp. 8 – 12.

101. Martin, R.C. (2000) "eXtreme Programming development through dialog", IEEE Software, 17(4), July-Aug. 2000, pp. 12 – 13.

102. Markus, M.L., (1983) "Power, Politics, and MIS implementation", Comm of ACM, 26(6), 1983, pp. 430 – 444.

103 McCauley, R., (2001) "Agile development methods poised to upset status quo", December 2001, ACM SIG Computer Science Education (SIGCSE) Bulletin, 33(4),

104. McCormick, M., (2001) "Technical opinion: Programming extremism", Comm. of ACM. 44(6), June 2001, pp. 109-119.

105 Mosemann, R., and Wiedenbeck, S., (2001) "Navigation and comprehension of programs by novice programmers", Program Comprehension, 2001. IWPC 2001. Proceedings. 9th International Workshop on, 12-13 May 2001, pp. 79 – 88.

106. Muller, M.M. and Tichy, W.F. (2001) "Case study: extreme programming in a university environment", Software Engineering, 2001. ICSE 2001. Proceedings of the 23rd International Conference on, 2001, pp. 537 – 544.

107. Myers, I. B. and Myers, P. B., (1995) Gifts differing: understanding personality type, Davies-Black Pub., 1995.

108 Nawrocki, J., Jasinski, M., Walter, B., and Wojciechowski, A., (2002) "Extreme programming modified: embrace requirements engineering practices", Requirements Engineering, 2002. Proceedings. IEEE Joint International Conference on, 9-13 Sept. 2002, pp. 303 – 310.

109 Nawrocki, J., Walter, B., and Wojciechowski, A., (2001) "Toward maturity model for extreme programming", Euromicro Conference, 2001. Proceedings. 27th, 4-6 Sept. 2001, pp. 233 – 239.

110 Niedrich, R. W., Sharma, S., and Wedell, D. H., (2001) "Reference Price and Price Perceptions: A Comparison of Alternative Models", Journal of Consumer Research, Dec., 2001, 28(3), pp. 339 - 354.

111. Nosek, J., (1998) "The Case for Collaborative Programming," Comm. Of ACM, March 1998, 41(3), pp.105 – 108.

112 Novak, T. P., (1995) "MANOVAMAP: Graphical Representation of MANOVA in Marketing Research", Journal of Marketing Research, Aug., 1995, 32(3), Issue 3, pp. 357 – 368.

113. O'Neil, H. F., Jr., Allred, K., and Baker, E. L. (1997) "Review of workforce readiness theoretical frameworks." Workforce readiness: Competencies and assessment. Mahwah, NJ: Erlbaum. 1997.

114. Online message forum on XP website: http://www.bad-managers.com/js.matt/com.maff.forum.ForumServlet?key=XPGen. Date retrieved, Nov-25-2003.

115 Palmer, S. R., and Felsing, M., A (2001) Practical Guide to Feature-Driven Development, 1st edition, Pearson Education, 2001.

116. Paulk, M., Curtis. B., Chrissis, M., and Weber, C., (1993) Capability Maturity Model for Software (Version 1.1), 1993, http://www.sei.cmu.edu/pub/documents/93.reports/pdf/tr24.93.pdf. Date retrieved, Nov-25-2003.

117. Pervin, L. A., (1989) Personality: Theory and Research, 5th ed. Wiley & Sons, Inc., 1989.

118 Peter, J. P., Ryan, M. J., and Hughes, R. E., (1975) "A MANOVA approach to disentangling correlated dependent variables in organizational research", Academy of Management Journal, Dec. 1975, 18(4), pp. 904 – 911.

119. Pressman, R.S., (1997a) Software engineering, Dorfman, M., and Thayer, R.H., (Eds.), IEEE Computer Society Press, Los Alamitos, CA, 1997.

120. Pressman, R.S., (1997b) Software engineering; a practitioner's approach, 4th ed., New York: McGraw-Hill, 1997.

121. Prowell, S., J., Trammell, C., J., Linger, R., C., and Poore J., H., (1999) Cleanroom Software Engineering, technology and process, Reading, MA. Addison-Wesley, 1999.

122 Reifer, D.J., Basili, V.R., Boehm, B.W., and Clark, B., (2003) "Eight lessons learned during COTS-based systems maintenance", Software, IEEE, 20(5), Sept.-Oct. 2003, pp. 94 – 96.

123. Rettig, M., (1994) "Prototyping for tiny fingers", Comm of ACM, 37(4), 1994, pp.21 - 27.

124. Revelle, W., website A, http://pmc.psych.nwu.edu/perproj/readings.html#readings. Date retrieved, Nov-25-2003.

125. Revelle, W., website B, http://pmc.psych.nwu.edu/perproj/theory/big3.table.html., Date retrieved, Nov-25-2003.

126. Riehle, D., (2001) "A Comparison of the value systems of adaptive software development and extreme programming: How methodologies may learn from each other", Extreme Programming examined, Succi, G., and Marchesi, M., (Eds.), Boston: Addison-Wesley, 2001.

127. Rising, L., and Janoff, N. S., (2000) "The Scrum Software Development Process for Small Teams," IEEE Software, July/August 2000 pp. 2 - 8.

128 Rosenthal, R., and Rosnow, R., L., (1991) Essentials of Behavioral Research: methods and data analysis, 2nd ed., McGraw-Hill, 1991.

128 Rowntree, D., (2003) Statistics Without Tears: A Primer for Non-Mathematicians, Pearson Publication, 2003.

129. Royce, W.W. (1987) "Managing the development of large software systems: concepts and techniques", Proceedings of the 9th International Conference on Software Engineering, 1987, pp. 328 – 338 http://www.acm.org/pubs/articles/proceedings/soft/41765/p328-royce/p328-royce.pdf. Date retrieved, Nov-25-2003.

130 Saiedian, H., (2003) "Panel: eXtreme programming: helpful or harmful?" Software Engineering, 2003. Proceedings. 25th International Conference on, 3-10 May 2003, pp. 718 – 718.

131. Scharer, L., (1997) "Pinpointing Requirements", Software Requirements Engineering, 2nd edition, Thayer, R.H., and Dorfman, M., (Eds.), Los Alamitos, CA.: IEEE Computer Society Press, 1997, pp. 30-35, 1997.

132 Schuh, P., (2001) "Recovery, Redemption, and Extreme Programming", IEEE Software, 18(6), pp. 34-41, November 2001.

133. Schwaber, K., and Beedle, M., (2001) Agile software development with scrum, Upper Saddle River, NJ: Prentice Hall, 2001.

134. SCRUM website http://www.controlchaos.com/, Date retrieved, Nov-25-2003.

135. Shaft, T. M., and Vessey, I., (1998) "The Relevance of Application Domain Knowledge: Characterizing the Computer Program Comprehension Process," Journal of Management Information Systems, Summer 1998, 15(1), pp. 51-78.

136. Shneiderman, B., (1998) Designing the user interface: strategies for effective human-computer-interaction, 3rd ed. Reading, Mass: Addison Wesley Longman, 1998.

137. Shneiderman, B., (1980) Software psychology : human factors in computer and information systems. Winthrop Pub., Cambridge, MA.

138. Smith, M. F. (1991) Software prototyping: adoption, practice, and management London; New York: McGraw-Hill, 1991.

139. Sommerville, I., (2000) Software Engineering 6th ed. Harlow, England; New York: Addison-Wesley, 2000.

140. Spitzberg, B. H., (2002) "Methods of Interpersonal Skill Assessment," The handbook of communication and social interaction skills. Mahwah, NJ: Erlbaum. 2002, pp. 93-134.

141. Spitzberg, B. H., (1997) A User's Guide to Relational Competence and Related Measures, unpublished., 1997.

142. Spitzberg, B. H., and Cupach, W. R. (1984) Interpersonal communication competence. Beverly Hills, CA: Sage, 1984.

143. Stalk., G, and Hout., T. M., (1990) Competing against time : how time-based competition is reshaping global markets, New York : Free Press ; London : Collier Macmillan, 1990.

144 Straub, D. W., (1989) Validating Instruments in MIS Research, MIS Quarterly, June 1989, 13(2).

145. Stephens, M., (2001) "The Case Against Extreme Programming," 2001, website http://www.bad-managers.com/Features/xp/case_against_xp.shtml. Date retrieved, Nov-25-2003.

146. Stewart, L. P., Cooper, P. J., Stewart, A. D., and Friedley, S. A., (1996) Communication and gender, 3rd ed. Scottsdale, AZ: Gorsuch Scarisbrick.1996.

147. Tannen, D., (1991) You just don't understand: women and men in conversation, New York: Ballantine Books, 1991.

148. Taylor, D., (1992) Global Software: Developing Applications for the International Market, New York: Springer-Verlag, 1992.

149. Thayer, R. H., (1997) "Software System Engineering: An Engineering Process" Software Requirements Engineering, 2nd edition, Thayer, R.H., and Dorfman, M., (Eds.), Los Alamitos, CA.: IEEE Computer Society Press, 1997, pp. 11-12, 1997.

150. Thompson, H. L., (2000) Introduction to the CommunicationWheel, Wormhole Publishing, 2000.

151. Von Mayrhauser, A., (1990) Software engineering: methods and management Boston, MA: Academic Press, 1990.

152. Wallace, D. R., and Ippolito, L.M., (1997) "Verifying and Validating Software Requirements Specifications", Software Requirements Engineering, 2nd edition, Thayer, R.H., and Dorfman, M., (Eds.), Los Alamitos, CA.: IEEE Computer Society Press, 1997, pp. 437-452, 1997.

153. Weinberg, G. M., (1998) The psychology of computer programming, Silver anniversary edition. New York: Dorset House Pub., 1998.

154 Williams, L. and Cockburn, A., (2003) "Agile software development: it's about feedback and change" Computer, 36(6), June 2003, pp. 39 – 43.

155. Williams, L., Kessler, R.R., Cunningham, W., and Jeffries, R., (2000) "Strengthening the case for pair programming", IEEE Software, 17(4), July-Aug. 2000, pp.19 – 25.

156. Williams, L., and Kessler, R. R., (2000) "All I really need to know about pair programming I learned in kindergarten" in Communications of the ACM, 43(5), May 2000, pp. 108 – 114.

157. Williams, L, and Kessler, R. R., (2002) Pair programming illuminated, Addison-Wesley, 2002.

158. Williams, L., and Upchurch, R., (2001) "In Support of Student Pair Programming," 2001 SIGCSE Conference on Computer Science Education, Charlotte, NC, February 2001. http://collaboration.csc.ncsu.edu/laurie/Papers/WilliamsUpchurch.pdf. Date retrieved, Nov-25-2003.

159. Williams, L., (2000) Collaborative Software Process, PhD Thesis 2000, http://www.cs.utah.edu/~lwilliam/Papers/dissertation.pdf., Date retrieved, Nov-25-2003.

160 Wiedenbeck, S. and Ramalingam, V., (1999) Novice comprehension of small programs written in the procedural and object-oriented styles, International Journal of Human-Computer Studies, 51(1), July 1999, pp. 71-87.

161 Wright, D. B., (1998) Understanding Statistics: An Introduction for the Social Sciences, Sage Publications, 1998.

162 Wright, T., and Cockburn, A., (2002) "Evaluating computer-supported collaboration for a problem-solving task", Computers in Education, 2002. Proceedings. International Conference on, 3-6 Dec. 2002, pp. 266 –267, Vol.1.

163. XP newsgroup; comp.software.extreme-programming.

164. XP website http://www.extremeprogramming.org. Date retrieved, Nov-25-2003.

165. Yahoo XP user group website A: http://groups.yahoo.com/group/xpusergroups/, Date retrieved, Nov-25-2003.

166. Yahoo XP user group website B: http://groups.yahoo.com/group/extremeprogramming., Date retrieved, Nov-25-2003.

167 Yourdon, E., (1997) Death March, Upper Saddle River, N.J. Prentice-Hall.