ABSTRACT

KATIRA, NEHA ANUPKUMAR. Understanding the Compatibility of Pair Programmers. (Under the direction of Dr. Laurie Ann Williams).

As pair programming is gaining broad acceptance in software engineering, programmers in academia and industry desire to work with compatible partners. Computer science instructors wish to proactively form student pairs to increase the likelihood of compatible pairs. This research investigates patterns that predict pair compatibility, and is aimed at improving the chances of forming mutually compatible pairs in academia and industry.

Female and minority students are under-represented in computer science. To make the programming experience of these under-represented groups more satisfactory, an analysis of the factors that possibly impact the compatibility of female and minority student pair programmers were conducted at the North Carolina State University.

A structured experiment involving 1053 undergraduate and 112 graduate students was carried out at the North Carolina State University to understand and predict pair compatibility. Additionally, 72 students were included later in the study to examine their work ethic and time management preferences. Three hundred and thirty nine industry pair programmers participated in the experiment by responding to a formal survey.

Analysis of the results indicates that students are more compatible with partners whom they perceive of similar skill level, although instructors cannot proactively manage these pairs. Pairing two female students in a pair is likely to result in a compatible pair in the undergraduate classroom, while mixed gender pairs are less likely to perceive compatibility. The greater the difference in the programming self-esteem of the undergraduate software engineering students, the more likely they are to be compatible, while the greater the

difference in the programming self-esteem of the graduate female students and their partner, the less likely it is that they perceive compatibility. The greater the difference in the midterm of pair (OO majority), the more likely it is that they perceive compatibility with their partner. As the difference in the GPA of the OO minority students with their partner increases, the less likely they are to perceive compatibility. We also observe as the difference in the SAT of the pair increases in the SE class, the less likely they are to perceive compatibility. The higher the GRE of the OO students, the lower they perceive their programming self-esteem.

Pair programmers in the industry perceive compatibility with a partner, who has a similar work ethic. A majority of the respondents believed that a significant benefit of pairing with a compatible partner is that the programming experience is more enjoyable, while a major setback of pairing with an incompatible partner is that it makes the programming experience less enjoyable.

UNDERSTANDING THE COMPATIBILITY OF PAIR PROGRAMMERS

By

NEHA ANUPKUMAR KATIRA

A thesis submitted to the Graduate Faculty in partial fulfillment
of the requirements for the Degree of

Master of Science in **COMPUTER SCIENCE**

Department of Computer Science

North Carolina State University

2004

APPROVED BY: _____

Dr. Laurie Ann Williams, Chairperson of Advisory Committee

_____

Dr. Edward F. Gehringer

_____

Dr. Jason A. Osborne

Date: _____

In loving memory of my parents – for their unconditional love and for teaching me that

***Success is a journey, not a destination*!**

To my nephew, Sidhant, whose smile fills cheer in my day.

BIOGRAPHY

Neha Anupkumar Katira earned her Bachelor of Science degree in Computer Science at Meredith College, Raleigh, North Carolina in August 2002. Her father's inspiration to *follow her dreams* led Neha to the Master of Science in Computer Science program at the North Carolina State University. At NCSU, under the guidance of her adviser, Dr. Laurie Williams, Neha worked on a conference paper, *Understanding the Compatibility of Student Pair Programmers*. What Neha started as a conference paper, evolved into her Master's thesis – *Understanding the Compatibility of Pair Programmers*.

ACKNOWLEDGEMENTS

I wish to thank Dr. Laurie Williams for her support and for believing in me. I would also like to thank my committee members, Dr. Jason Osborne and Dr. Edward Gehringer for their suggestions and guidance during the course of my thesis. Thank you to Dr. Thomas Honeycutt for his inspiration. Thanks to Xiao Ni for his recommendations on the data analysis. I appreciate the cooperation of Carol Miller, Suzanne Balik, and Dr. Edward Gehringer for their cooperation in having their classes participate in this research. This material is based upon work supported by the National Science Foundation under Grant No. 00305917. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

I thank all my family and friends for always being there for me.

TABLE OF CONTENTS

LIST OF TABLES

Page

LIST OF FIGURES

Page

1. INTRODUCTION

Pair programming (Williams and Kessler 2003) is a style of programming in which two programmers work side-by-side at one computer, collaborating on the same design, algorithm, code, or test. In pair programming, one programmer is the *driver*, controlling the input devices such as the keyboard and the mouse, producing the design or the code. The other programmer is the *navigator*, who actively and continuously examines the driver's work. There are numerous benefits of pair programming over solo programming (Cockburn and Williams 2001) including continuous code review, effective problem solving, more learning, programmer satisfaction, sharing of knowledge, improved design quality, and effective communication between members of the team.

Research has shown that students who pair program perform at least as well as students who program solo. Pair programming students are more likely to continue with further computer science classes and to major in computer science. (Bevan, Werner et al. 2002; McDowell, Werner et al. 2002; McDowell, Werner et al. 2003; Nagappan, Williams et al. 2003; Nagappan, Williams et al. 2003; Williams, McDowell et al. 2003) Further, evidence suggests that students involved in collaboration outside of class are more likely to do well in class exams than students who work solo. (Joseph, Payne et al. 2003)

When pair programming, most often, pairs are compatible and have productive working relationships. Occasionally, though, programmers are not compatible with their partners. When pairs are not productive, the benefits discussed above may not be realized. *The objective of this research is to determine factors that influence pair compatibility to improve*

the chances that a programming pair will be mutually compatible. In this research, pair compatibility is studied in both academia and in industry.

To analyze the factors that influence the compatibility of pair programmers in the computer science classroom, an experiment involving 1,053 undergraduate and 112 graduate students at North Carolina State University (NCSU) was carried out primarily during the fall 2002, spring 2003, and fall 2003 semesters. The courses in which the research was performed are freshman Introduction to Programming – Java (CS1), undergraduate (junior/senior) Software Engineering (SE), and graduate Object-Oriented (OO) Programming. After each pairing cycle, the students were required to complete a web-based peer-evaluation survey about the contributions of their partner, and how compatibly the pair worked together. This survey is discussed in Section 3.3. Data for this research were obtained from the answers to the compatibility questions on the survey. Additionally, 72 SE students were studied in the fall 2004 semester to examine the impact of the students' work ethic and time management preferences on pair compatibility. Factors affecting the compatibility of pair programmers in the industry were studied through responses to a web survey, as will be discussed in Section 5. The survey was completed by 339 respondents.

The decreasing numbers of female and minority (African-American and Hispanic) students in computer science (Statistics 2002) has been a concern for many years. A paradigm shift from solo programming to a collaborative style of programming may make a career in Information Technology (IT) more attractive to women and minorities. These students may be more productive in team environments, and pair programming may help alleviate their concern of the lack of social interaction stereotypical of IT positions. Pair

programming has the potential of increasing the retention rates of female and minority in IT education. As a result, we performed targeted analysis on the compatibility of women and minorities in the CS1, SE, and OO classes.

The eight hypotheses of the study are as follows:

Pairs are more compatible if students with…

- different **Myers-Briggs personality types** are grouped together.
- similar **actual skill level** are grouped together.
- similar **perceived skill level** are grouped together.
- similar **programming self-esteem** are grouped together.
- same **gender** are grouped together.
- similar **ethnicity** are grouped together.
- similar **work ethic** are grouped together.
- similar **time management** preferences are grouped together.

The remaining chapters are as follows: Chapter 2 describes a brief overview of the concepts that this research is based on, including pair programming, Myers-Briggs personality type, actual skill, self-esteem, perceived skill, gender, and ethnicity. Chapter 3 gives the research methodology and procedures used for the data collection and the analysis of the academic study. Chapter 4 provides the results of the student compatibility data analyses. Chapter 5 outlines the description and the results of the survey that was used to

study the factors influencing the compatibility of pair programmers working in industry. Chapter 6 gives a summary of the work and the significant findings of the research.

2. BACKGROUND

In this chapter, significant contributions in the research of pair programming, and other related work to the thesis are explored. Section 2.1 outlines the pair programming paradigm and other work in the field, followed by sections on Myers Briggs personality type, pairing and preferences in skill level, pair programming and self-esteem, pair programming and perceived skill, pair programming and gender, and finally pair programming and ethnicity.

2.1 Pair Programming

Pair programming (Williams and Kessler 2003) is a style of programming in which two programmers work side-by-side at one computer, collaborating on the same design, algorithm, code, or test. Pair programming has been used by programmers sporadically for many years. Recently, attention has been focused on pair programming as one of the key practices of the Extreme Programming (XP) (Williams and Kessler 2003) software development methodology. In pair programming, one programmer is the *driver*, controlling the input devices such as the keyboard and the mouse, producing the design or the code. The other programmer is the *navigator*, who actively and continuously examines the driver's work. The role of the navigator is to point any defects, suggest alternatives, look up resources, ask questions, and consider the design implications. Through this process, the navigator identifies any tactical and strategic defects in the design or code. Tactical defects include syntax errors, typos, invoking the incorrect method name, etc. Strategic defects are identified by the navigator when the driver is perceived to be headed down the wrong path, i.e., when what they are implementing may not achieve the desired result. The navigator plays the role of a strategic thinker and thinks strategically about the direction of the work.

In addition, the driver and the navigator can brainstorm on-demand at any time. During their collaboration, the driver and the navigator discuss the direction of the program, the best choice of algorithm, or anything that may assist in solving the problem. This brainstorming is an asset to the pair in finishing in a faster and more effective manner. (Beck 2000)

The driver and the navigator switch roles periodically, allowing them to learn and practice new skills, and contribute more to the work. There are no rules as to how often the roles be switched between the driver and the navigator. However, switching roles should occur at a natural transition point in the design or coding phase. In organizations, where there are more than one pair, pairs may rotate partners after the completion of a task or a subtask. Pair rotation allows programmers to learn a variety of aspects of the system, and learn skills from different partners.

The workplace facilities and furniture may also support pair programming. The most common and recommended configuration is to use one monitor, one keyboard, and one mouse for the pair. Pairing programmers sit side-by-side so that each has a clear view of the monitor, facilitating the transfer of control within the pair by sliding the keyboard and the mouse. In certain situations, the configuration of two monitors, two keyboards, and two mice may be appropriate. (Williams and Kessler 2003)

2.1.1 Benefits of Pair Programming

There are numerous benefits of pair programming over individual solo programming. The proverbial phrase *two heads are better than one* describes pair programming (Williams, Kessler et al. 2000). Below is a list of the benefits (Cockburn and Williams 2001) of pair programming:

- *Continuous Reviews*: By the virtue of being under a constant observation of two programmers, the design and code is continuously reviewed, thereby potentially eliminating defects earlier (as compared to the work of solo programmers).
- *Problem Solving*: The combined thinking of the two brains has the possibility of solving difficult problems at a faster rate than if it were tackled by a solo programmer. Programmers share their myriad of knowledge and energy in solving the problem.
- *Learning*: Programmers learn skills from each other. Pairing partners take turns being the teacher and the taught, from moment to moment switching roles. Coding standards, programming language rules, and design and programming idioms are learned and followed.
- *Satisfaction*: Pair programming teams that had earlier programmed alone, report enjoyment in pair programming and they show more confidence in their programs.
- *Sharing of knowledge*: Pair programming has the benefit that more than one person is aware of the system that is being created, thereby reducing the overhead and risk caused by the loss of a valuable staff member.
- *Design Quality*: When programmers pair in teams and brainstorm together on the design, the program generally ends up well designed when compared to the work of a solo programmer.
- *Communication*: Programmers on the team share both the problems and the solutions, and are less likely to have hidden goals from each other, thereby increasing the

information flow within the team. Communication results in increased team effectiveness. (Palmieri 2002)

These numerous benefits of pair programming (Cockburn and Williams 2001) encourage the use of pair programming in organizations.

2.1.2 Prior Research in Pair Programming

Formal research in pair programming started in 1998. Nosek's study (Nosek 1998) of 15 experienced pair programmers reported that pairs required less elapsed time working on the program, and that the pairs produced better code than the individuals. In the experiment, the programmers worked up to 45 minutes on a complex problem related to their work. Five of these programmers worked individually, and 10 programmers worked in pairs in the experiment. The five solo programmers formed the control group in the experiment. Conditions and resources were identical for the solo and the pair programmers. The pair programmers spent 43% more total time on the task than the solo programmers, but because they worked in tandem, the pairs completed the task in 40% less time than the solo programmers. The average completion time for the individual programmers was 42 minutes, while that of the pairs was 30 minutes. Nosek measured the impact of pair programming based on five independent variables: time, functionality, readability, programmer confidence, and enjoyment. Scores were better for pairs in all the measures, but the differences in the time and readability scores were not statistically significant. The combined scores of the measured variables was about 35% better for the pair programmers than the solo programmers. (Williams 2000)

In 1999, Williams (Williams 2000) conducted an experiment involving 41 senior computer science students at the University of Utah. The students were enrolled in a software engineering class. Four programming assignments were completed throughout the semester. In the study, students were split in two groups, composed of students with a range of similar grade point averages. The students were split in 14 pairs and 13 individual programmers. The results indicate that the pairing students wrote higher-quality code than the individual students. The code developed by the pairs passed 15% more of the teaching staff's automated test cases as compared to the code of the individual programmers. The pairs took 15% more person-hours to complete the assignments than the individual programmers. However, this time increase was not statistically significant. The results indicate that pair programming has the potential to produce higher quality code in relatively the same amount of time as solo programming. (Williams 2000)

A family of pair-programming experiments encompassing over 1,200 beginning computer science students was conducted at the University of California at Santa Cruz (UCSC) and NCSU. (Williams 2000; Bevan, Werner et al. 2002; McDowell, Werner et al. 2002; McDowell, Werner et al. 2003; Nagappan, Williams et al. 2003; Nagappan, Williams et al. 2003) The results suggest that students who pair programmed performed at least as well on exams as the students who programmed independently. Pair programming students were also more likely to continue with further computer science classes and to major in computer science. The results of an experiment conducted at NCSU (Williams, McDowell et al. 2003) show that pair programming creates a laboratory environment conducive to students being more productive and less frustrated. Further, a study (Joseph, Payne et al.

2003) of undergraduate students in a computer architecture course at Pace University in New York reports that students involved in collaboration outside of class are more likely to do well in the class exams.

2.1.3 Human Cooperation

Vogel (Vogel 2004) raises some very interesting questions in her article regarding human cooperation: *Are cooperative urges ingrained in our genes? Or are we taught by our culture to play well with others?* Vogel presents the example of an experiment called the "Trust Game," where one player, the "truster," is given some money. The truster then decides if he or she wants to share any part of that money with an anonymous partner, the "trustee." According to the rules of the game, the trustee receives double the amount of money donated by the truster, and can decide how much money to give back. Results of multiple such experiments show that more than half of the trustees return some of the money, and they tend to transfer more money when entrusted with more money. Observation of the game shows that when players carry their cooperation history with them through repeated rounds of games (instead of remaining completely anonymous), cooperation by their partners increases dramatically – from about 50% to more than 80%. A player builds his/her cooperation history when he/she is willing to give a big share of the money to the trustee, who may not pay them back, but other people see the charitable act. Players who have a history of cooperation in the game are much more likely to receive cooperation from future partners. Additionally, even when players know that their identity will be kept secret and are told that they will never encounter their partner again, up to 50% of the players still cooperate in the game (Vogel 2004). The notion that working compatibly with a partner will yield the

students a good reputation seems familiar with the idea resonating in the trust game. When a student is compatible with a partner, he or she will gain a good reputation with other classmates when pairing in future class assignments.

Describing the results of the Functional Magnetic Resonance Imagery (fMRI) of people playing the trust game, Vogel writes that after the game, seeing faces of cooperative partners triggered activity in brain areas linked to social cognition and reward. This result was concluded by the Wellcome Department of Imaging Neuroscience in London. In addition, participants were given an opportunity to punish partners who did not cooperate. The fMRI scan results revealed that the idea of punishing activates similar kinds of reward-related pleasure circuits as does eating sweets, called as the "sweet taste of revenge" (Vogel 2004). These findings suggest that if two people are to form a team, their partnership will likely succeed if the reward is as important to one partner as the other. Rewarding and punishing partners in this research was facilitated through the peer evaluation survey, as discussed in Section 3.3. Students could affect their partner's grade in class through their peer evaluation.

2.2 Myers-Briggs Personality Type

The Myers-Briggs Type Indicator (MBTI) is popularly used to classify people into one of the sixteen personality types. The MBTI stems from Carl Jung's theory of personality types (Keirsey 1998) and measures human preference on four bipolar dimensions. Jung's theory (Jung 1971) suggests that variation in human behavior is not due to chance, but the variation is attributed to basic and observable differences in the ways people prefer to use their minds to gather and process information. The four dimensions include

introversion/extraversion (EI index), intuition/sensing (SN index), feeling/thinking (TF index), and judgment/perception (JP index).

- The EI index indicates *extraversion*, an orientation toward the outer world, focusing on people and things, and *introversion*, an orientation toward the inner world of ideas.
- The SN index measures perception, i.e., the means by which one becomes aware of people, things, events, and concepts. *Sensing* perception uses the physical senses of seeing, hearing, tasting, touching, and smelling, while *intuitive* perception uses an intangible, usually unconscious sense.
- The TF index measures judgment, i.e., the means of coming to conclusions about how to handle the information gathered. *Thinking* judgment involves making decisions objectively, based on laws, principles, and information. *Feeling* judgment makes decisions subjectively and personally, based on relationships and values.
- The JP index measures the individual's preference in dealing with the outer world. J indicates a preference for using a *judgment* process (thinking or feeling), while P indicates a preference for using a process of *perception* (either sensing or intuition).

All four indices are dichotomous, as people tend to develop one preference on the scale at the expense of the other (Jung 1971). An MBTI personality type consists of a four-letter code, such as ESTJ (Extraverted Sensing Thinking Judging), to indicate the personality type of the individual. All possible combinations yield sixteen different personality types, each with a distinct descriptive profile of characteristic behavior patterns (Keirsey 1998).

Researchers (Keirsey 1998) hypothesize that people with S or T in their personality type are more likely to perform better in computer programming than people with N or F

respectively. Research (Bishop-Clark and Wheeler 1994) involving a pilot study of 34 introductory programming class students was conducted at a medium-size midwestern university, with a follow-up study of 114 college students attending a different campus of the university. The students in the follow-up study were taking the same computer programming class as the pilot study students. The students' cognitive characteristics of perception (S/N index) and processing information (T/F index) were considered for this research. The results reported that sensing students performed better than intuitive students, and judging students performed better than perceptive students on programming assignments, as evaluated by their assignment grade. The reason suggested is that sensors prefer to work with observable facts, tend to be focused toward practicality, have a memory for details, and prefer concrete, tangible experiences. However, the personality type of the students did not influence their test achievement or overall achievement. The results suggest that there are different sets of cognitive skills in writing a program versus taking a test. The researchers also concluded that personality type does not appear to be an important factor in predicting whether a student will drop a class (Bishop-Clark and Wheeler 1994).

David Keirsey (Keirsey 1998) conjectures that people with different personality types have different sets of strengths and weaknesses. The strengths and weaknesses of different types complement each other. Keirsey writes that basic differences are all around us. Seeing others as different from ourselves, we often conclude that differences are bad, and that people are acting strangely because something is unusual with them. Keirsey maintains the typology developed by Myers-Briggs, but groups the sixteen types into four categories: Rational (NT), Idealist (NF), Artisan (SP), and Guardian (SJ).

The personality types in the NT group are INTJ, INTP, ENTJ, and ENTP, while the NF group consists of INFJ, INFP, ENFJ, and ENFP. The SP group includes the personalities ISTP, ISFP, ESTP, and ESFP, and finally, the personality types in the SJ group are ISTJ, ISFJ, ESTJ, and ESFJ. Keirsey has added four kinds of intelligence (tactical, logistical, diplomatic, and strategic) to the Myers-Briggs framework, with each of the personality types having a different balance of these intelligence abilities. The Keirsey framework encourages a combination of differing strengths within a team to form a more effective whole. Keirsey suggests that it is wiser to accept the differences than to press for a "one size fits all" ideal way. Keirsey's theory is the basis of this research's hypothesis that pairs are more compatible if students with different personality types are grouped together.

2.3 Pairing and Preferences in Skill Level

Melnik and Maurer (Melnik and Maurer 2002) performed a survey-based study at the University of Calgary and Southern Alberta Institute of Technology (SAIT) in Canada. The survey was completed by students in different levels in their education ranging from second year undergraduates to graduate students with several years of software development experience. The researchers reported that students preferred working with a partner who matched their own educational qualifications and experiences. If the programming skill and experience of the students in the pair were not a match, the stronger partners tended to complain of an unbalanced share of the work. On an open-ended survey question, some of the stronger partners also suggested that such pairs are more productive when the stronger of the pairs is the driver. (Melnik and Maurer 2002).

Russian psychologist Lev Vygotsky's theory (Vygotsky 1978) can explain students' preference toward working with partners of similar skill level. Vygotsky defined the "Zone of Proximal Development" (ZPD) as the range of ability of a student with and without the assistance of a teacher or a more capable peer. The teacher or more capable peer could be an adult or another peer who has already mastered that particular function and who provides non-intrusive intervention. The lower end of the ZPD is the student's ability without assistance; the higher end of the ZPD is the student's ability with assistance (Doolittle 1997). The actual developmental level of a person includes all the functions that the person can perform on his or her own, independently without the help of anyone else, while the ZPD includes all the functions that a person can perform only with the help of someone else. Vygotsky emphasizes the importance for the teacher/peer to remain within the student's ZPD. If the teacher/peer works too far above the student's ZPD, the student will become confused, and no intellectual growth will occur. If the teacher/peer works too far below the student's ZPD, the student will not be challenged enough and will stagnate. Accordingly, pair-programming students would learn the most from a partner who operates within their own ZPD. Vygotsky's theory stresses that learning is a social process that does not happen by itself but instead occurs through interaction with others. (Doolittle 1997)

2.4 Pair Programming and Self-esteem

A survey-based (Thomas, Ratcliffe et al. 2003) study involving more than 60 students at the University of Wales examined the correlation between self-reported programming self-esteem and the performance of first-year students in an introductory programming course that

used pair programming. The study asked the students to rate themselves on a scale of 1-9 on a programming attitude questionnaire, as follows:

1 – (Code-a-Phobe): I don't like programming, and I think I am not good at it. I can write simple programs, but have trouble writing new programs for solving new problems.

9 – (Code-Warrior): I have had no trouble at all completing programming tasks to date, in fact they weren't challenging enough. I love to program and anticipate no difficulty with this course.

Code-a-phobe defines those students that decide to major in computer science but do not obtain much satisfaction programming. Additionally, they often report hating programming. (Weber-Wulff 2000) The results of the survey-based study revealed that students with lower programming self-esteem liked pair programming more than students with higher programming self-esteem did. There was also some evidence that high self-esteem students disliked pair programming the most when paired with a low self-esteem partner. Additionally, students seemed to do their best work when paired with students of similar levels of programming self-esteem. (Thomas, Ratcliffe et al. 2003)

2.5 Pair Programming and Perception of Partner

The process of forming a perception of others is achieved through the mechanism of self-comparison with others. Self-comparisons with others involve processes that are likely to have self-evaluative consequences. Specific knowledge of the self is accessible easily, hence it is used while forming perceptions of others. Mussweiler and Bodenhausen (Mussweiler and Bodenhausen 2002) write that social comparisons increase the specific knowledge about the self, "… participants evaluated themselves to be less competent if they were exposed to a

super-competent other" (Mussweiler and Bodenhausen 2002). In our research, students were asked for their perception of their partner's skill level relative to themselves on a Likert Scale (Better than me, About the same, Weaker than me).

Shaw and Steers (Shaw and Steers 2001) explain human perception in their research in social psychology. They conducted a study involving 270 participants with equal numbers of male and female participants at California State University, Northridge. These participants formed a likeability impression of a target male or female person. In the experiment, the participants were given 20 cards with appearance and trait information of the target. The trait information included attributes, such as personality traits, behavior, and demographic information. Appearance and trait information were presented to the subjects. Participants were told that the information cards in front of them contained information about a person and whether the person was a male or a female. The participants could then acquire information about the target person by selecting a card from either the personality or the trait stack. The participants repeated the process until they had formed an impression. After forming an impression, the participants were asked two questions. They were asked how much they liked the person on a scale from strongly dislike (1) to strongly like (7). They also indicated how confident they were in their decision on a scale from very unconfident (1) to very confident (7). The researchers conclude that the participants sought far less appearance information than any of the other traits. However, pairs of different genders accessed more appearance information, while same gender pairs accessed more trait information. Additionally, male perceivers accessed more appearance information about female targets than did any other perceiver-target gender combination. When a target person's intelligence

was made apparent along with the appearance, perceivers formed their impressions based on both types of information rather than being influenced by physical appearance alone (Shaw and Steers 2001). Forming the perception of a target using the target's appearance and/or trait information is related to this thesis since the subjects in this study were asked to evaluate their perception of their pairing partner's skill level and their perception of how compatibly they worked as a pair.

2.6 Pair Programming and Gender

The representation of women in computer science has been low (Statistics 2002). Tracy Camp (Camp 1997) reports that although the percentage of bachelor's degrees awarded to women in science and other engineering disciplines have increased, the corresponding percentages of computer science have decreased.

A few studies (Beyer, Rynes et al. 2003; Cohoon 2003) identify the reasons why women are a minority in computer science. Margolis and Fisher (Margolis and Fisher 1997) assert that the attractiveness of the IT field may be a problem with female students. Women frequently link their computer science interest to a larger societal framework. Margolis and Fisher conducted a longitudinal study following the students' experiences for two to four years including 51 female and 46 male computer science majors at Carnegie Mellon University. They found that the cultural stereotype of the successful computer science student is someone who is at the computer "twenty-four hours a day/seven days a week ('24/7'), living, thinking, and breathing computer science." Female students are concerned that their study of computer science may require a myopic focus of the computer, may detach them from people, and could require 24/7 dedication. Females want to "do something useful

for the society". Margolis and Fisher express their concern that many women students are hesitant to join the computer science world, where they fear that the links to other interests in their lives will disappear gradually. Twenty percent of the female students interviewed, questioned their choice of major in computer science because they felt that they did not share the same type of focus and intensity of interest they see in their male peers (Margolis and Fisher 1997).

Fisher, Margolis, et. al. (Fisher, Margolis, et. al. 1997) observed a gap between women's perceived ability and their actual performance in computer science classes. They note that first-year female students report themselves as being significantly lower in computing experience and in their ability to master the course material compared with the male students. (Fisher, Margolis, et. al. 1997)

Rayman and Brett (Rayman and Brett July/August 1995) suggest that socio-psychological issues such as self-confidence, perceived ability, and resiliency have been linked to female persistence in science. They further explain that research indicates that women attribute their success to extrinsic factors, such as luck, rather than intrinsic factors, such as ability. Rayman and Brett studied 369 women, categorizing them into groups of those who left the sciences after graduation (Leavers), those who switched to some other occupation at some point after college (Changers), and those who remained in the sciences (Stayers). Stayers were the students most likely to have received encouragement by both their teachers and their parents, especially their mothers, to pursue a career in science. Analyzing the factors that motivate female students, Rayman and Brett write, "Affiliation

activities such as advising, encouragement, and mentoring have been noted as playing a role in women's persistence in science" (Rayman and Brett July/August 1995).

The presence of fewer women with bachelor's degrees in computer science (Statistics 2002) implies that there will be fewer women in industry and academia. Too few women with doctoral degrees in computer science leads to few women faculty members, further implying an inadequate number of same-sex role models for female students (Nelson 1996). The low numbers of women with computer science education leads to low numbers of women in the professional work force. Cohoon (Cohoon 2003) emphasizes reasons for the need of the participation of women in computer science, including maintaining a supply of computer science professionals, providing diverse viewpoints, and promoting gender equity.

2.7 Pair Programming and Ethnicity

African-American and Hispanic students are underrepresented in the computer science classroom (Statistics 2002). Members of under-represented groups are likely to bear the impact of stereotypes. Claude Steele's research on "stereotype threat" (Steele 1997) offers an explanation for the link between the stereotypes of the negative expectations of the academic performance, a drop in confidence, and declining interest in their field of study. Steele writes that when one is in a situation in which a negative stereotype about one's group applies, one is fearful of confirming the stereotype, creating a stereotype threat resulting in poor performance and not identifying the self with the field. These students develop the fear of confirming stereotypically negative expectations of their performance (Steele 1997).

Research has shown that African-American success rates in science courses can be significantly improved by shifting the classroom learning paradigm from individual study to

one that capitalizes on group processes, such as student work groups and student-student tutoring. (Nelson 1996) Treisman (Treisman 1992) performed an experiment with African-American calculus students at Berkeley. He reported that about 60% of the African-Americans who had completed calculus at Berkeley in the preceding decade received grades of D or F, grades so low that they could not proceed with a major in mathematics, science, or technology. Prior social norms had taught these students that they needed to study alone and that working with others was considered cheating. In contrast, the most successful Asian-American students formed "study-squads" to get through calculus, groups in which social status was increased by one's ability to help others. In his experiment, Treisman required his students do peer checking and work in small, collaborative groups. Only about 4% of the African-American students completing Treisman's "workshop" calculus made a D or F, compared to the 60% before. Moreover, the differences vanished between the average grades achieved by the African-American students who did their workshop calculus, and grades achieved on the same exams by students in socially-dominant groups, including Asian-Americans. (Treisman 1992) These studies encourage us that minority students are more likely to succeed in class when they use pair programming.

3. STUDENT RESEARCH METHODOLOGY

During the fall 2002, spring 2003, and fall 2003 semesters, a formal experiment was conducted in the freshman Introduction to Programming – Java (CS1), the undergraduate Software Engineering (SE), and the graduate object-oriented (OO) programming classes. Students pair-programmed for their programming assignments in these classes. Students then evaluated their partner after each pairing assignment by completing a survey conducted through a web-based pair-programming management application (as will be discussed in Section 3.3). The survey required the students to evaluate the contributions of their partner for the assignment. For the purposes of this research, the students were asked two additional questions. These two questions were designed to elicit the students' perception of their partner's skill level and their joint compatibility as a pair. Data for this research on the compatibility of the students were obtained from the answers to these two questions.

The different factors that this research focuses on are: the Myers-Briggs personality type, actual skill measured by multiple measures (class midterm grade, SAT/GRE score, overall GPA), self-reported programming self-esteem, and the partner-perceived skill level.

The objective of this research is to determine factors that influence pair compatibility to improve the chances that a programming pair will be mutually compatible. The eight hypotheses of the study are as follows:

H1: Pairs are more compatible if students with different **Myers-Briggs personality type** are grouped together.

H2: Pairs are more compatible if students with similar **actual skill level** are grouped together.

H3: Pairs are more compatible if students with similar **perceived skill level** are grouped together.

H4: Pairs are more compatible if students with similar **programming self-esteem** are grouped together.

H5: Pairs are more compatible if students with same **gender** are grouped together.

H6: Pairs are more compatible if students with similar **ethnicity** are grouped together.

H7: Pairs are more compatible if students with similar **work ethic** are grouped together.

H8: Pairs are more compatible if students with similar **time management** preferences are grouped together.

Correlations were also computed to find an association between the partner-perceived skill and the actual skill of the students, and the programming self-esteem and the actual skill of the students.

Hypotheses 1-4 were a part of an initial study (Katira, Williams et al. 2004). The results indicated the students' perception of their partner's skill level has a significant influence on their compatibility. Graduate students work well with partners of similar actual skill level, measured by the midterm grades in class, and freshman students seem to work well with partners of different Myers-Briggs personality type. The programming self-esteem of the students does not seem to be a factor in predicting their compatibility. (Katira, Williams et al. 2004)

This chapter is organized into four sections. Section 3.1 presents the distribution of the students in the different classes, while Section 3.2 describes the pairing of the students for the

programming assignments. Section 3.3 outlines the peer-evaluation survey completed by the students, and Section 3.4 describes the process of the analysis of the data.

3.1 Students and Courses

A pair-programming lecture was presented to all students and teaching assistants (TAs) in the study. Students included in the study signed a consent form authorized by the NCSU Institutional Review Board. The consent form allowed the research team to utilize personal academic information, such as grade point average. The following subsection describes the students and courses in the study. Table 3.1 summarizes the total number of students, female, and minority students in the CS1, SE and the OO classes.

Table 3.1: Student Distribution in Classes

Class	No. of Students	No. of Female	No. of Minority
CS1	804	112	116
SE	249	42	22
OO	112	24	N/A

There were 23 students in the CS1 and seven students in the SE classes respectively who were both female and minority. The data from these students were not analyzed separately because of the low numbers. Therefore, they are listed in both columns (Female, Minority). The detail of the student distribution is outlined in the following sub-sections.

3.1.1 CS1: Introduction to Programming – Java

A total of 234 and 570 students were enrolled in the Spring 2003 and Fall 20003 classes, respectively. Of the 804 students, 112 were female students, and 116 students were minority. The students had two hours of lecture time and one three-hour closed lab each week. There were 26 lab sections with a maximum of 24 students in each lab section. During the lab time, the students completed a structured pairing assignment. The students had to complete four

assignments during the semester. The TA assigned the students a new partner at the completion of each assignment. The compatibility data of these 804 students for four partner cycles is analyzed in this research.

3.1.2 SE: Software Engineering

In the fall 2002 semester, 143 students were enrolled in the SE class, while 106 students were enrolled during fall 2003. Of the 249 students, 42 were female students, and 22 students were minority. Each semester, the class had two hours of lecture time and two hours of closed lab weekly. The students had multiple two- to three-week programming assignments. Students were mandated to work in pairs, with different assigned partners for the assignments, overseen by the TAs. Each class ended with a three- to five-person, six-week team project. The compatibility data from these 249 students for the assignments over the two semesters is analyzed in this research.

After most of the data for this research had been collected, in Fall 2004, we asked the 136 students enrolled in the SE class to respond to questions related to their work ethic and time management preferences. These data were used to compute a correlation between these new factors (work ethic and time management) and pair compatibility.

3.1.3 OO: Object-Oriented Languages and Systems

The graduate class offered in Object-Oriented Languages and Systems had 62 and 50 students in the fall 2002 and fall 2003 semesters, respectively. The class required students to work on three class projects. There were no lab sections associated with the class. Students had the option to work in pairs or work solo; 37 students in fall 2002 and 45 students in fall 2003 chose to pair for at least one assignment. The students signed up on a message board

for pairing suggesting the name of preferred partners, if they chose to pair. The class TA then paired the students for the project. There was no moderated pair programming in the OO class. The compatibility data of these 112 graduate OO students was analyzed over three pairing cycles. Ethnicity information was available only for the fall 2003 class.

3.2 Pairing Methodology

The assignment of pairs in the CS1 and the SE classes was identical due to the facility of closed labs, where pairing was moderated by the TAs. The OO students had the option to work solo or to work in pairs.

To examine factors that might influence pair compatibility, pairs were formed in the CS1 and the SE classes based on certain criteria as outlined in the hypotheses. At the beginning of the semester, students reported their Myers-Briggs personality type. For one of the pairing cycles, students were paired with a partner based on an exact match on the personality type (all four letters). When an exact match for the students was not available, students were randomly paired with a partner.

For a different pairing cycle, students were paired with a partner who had a similar midterm grade in the class. A similar midterm grade is defined within a range of +/– 10 points of the grade of the student.

Additionally, students were required to assess their programming self-esteem, as discussed in Section 2.4, on an ordinal scale of 1–9 with the following end-points:

1 = I don't like programming and I think I am not good at it. I can write simple programs, but have trouble writing new programs for solving new problems.

9 = I have no problems at all completing programming tasks to date, in fact they weren't challenging enough. I love to program and anticipate no difficulty with this course.

Although there was no pairing scheme that paired students with similar self-esteem together, this data was used to analyze the compatibility data of the pairs. Similarly, gender and ethnicity demographics and the perceived skill of the students were used in the analysis of the compatibility of the students.

3.3 Peer evaluation

When students work in pairs, computer science instructors may be concerned whether each student does his or her fair share in the completion of the project. Students may be concerned that their assigned partner may not do a fair share of the work. To address these concerns, students in the study were asked to complete a web-based peer evaluation survey at the end of each pairing assignment. Students understood that answers to the peer evaluation survey on the contribution questions could affect their course grades.

Students assigned 0 – 20 points on each of the following five questions, which gave their partner a score of 0% – 100% on their peer evaluation:

1. Did the student read the lab assignment and preparatory materials before coming to the scheduled lab?
2. Did the student do his/her fair share of work?
3. Did the student cooperatively follow the pair-programming model (rotating roles of driver and navigator)?
4. Did the student make contributions to the completion of the lab assignment?
5. Did the student cooperate?

For the purposes of this research, students were asked two additional questions, which were no-penalty questions. These two questions were designed to elicit the students' perception of their partner's skill level and their joint compatibility on an ordinal scale:

1. Assess the technical competency of your partner relative to yourself. (Better than me, About the same, Weaker than me).
2. Assess how compatible you and your partner were. (Very compatible, OK, Not compatible).

3.4 Data Analysis

The responses of the students on the peer evaluation survey were stored in a MySQL database. The responses were then extracted to an Excel file for analysis. The data was cleaned. A major task in the data-cleaning process was to sort the data manually to organize data of partners for a pairing assignment in consecutive rows. Then for each student, all the data including the Myers-Briggs personality type, SAT/GRE score, midterm grade, grade point average (GPA), self-reported subjective measure of self-esteem, gender, ethnicity demographics, the partner perceived skill level, and compatibility had to be recorded on the same row in the Excel file.

After the data-cleaning process, code was written in SAS to analyze the data. PROC GENMOD in SAS was used for the multivariate analysis of the data. Logistic regression with the multivariate analysis identifies the principal factors that contribute to the compatibility of the pair. PROC GENMOD allowed the data to be fit in a Generalized Linear Model. The Generalized Linear Model is used for the multiple regression of several independent predictor variables and one response variable. Compatibility perceived by the

students is the response variable in the study. The basic assumption behind the Generalized Linear Model is that that the data is not normally distributed; the Generalized Linear Model allows for other distributions than normal.

An advantage of using the Generalized Linear Model is that the repeated subject measure can be analyzed for the response variable. In this research, compatibility responses were analyzed for individual students and also for pairs (made up of two individual students). With compatibility responses of individual students, the factors that affect the compatibility of individual students were analyzed. With the compatibility responses of pairs, the factors affecting mutual pair compatibility were analyzed (e.g., did both students report compatibility, did only one report compatibility, did neither report compatibility).

For the pair-level analysis, each pair was uniquely identified by a unique pair identifier (pairid). The NCSU computer network userid of each of the students in the pair was combined to form the pairid. For example, "jones" and "smith" were combined to form "jonessmith." Then the combined pair data was recorded on the row for the pair. The pair data included:

- pair-personality (1 for similar or 0 for different),
- pair-SAT/pair-GRE score (absolute difference of scores),
- pair-midterm (absolute difference of scores),
- pair-GPA (absolute difference of scores),
- pair-self-esteem (absolute difference of scores),
- pair-gender (MM or MF or FF where M is male, F is female),

- pair-ethnicity (MM or NM or NN where M in minority and N is non-minority),
- pair-perceived skill (absolute difference of scores and average of the scores). For example, if student A perceived his partner's skill level as 1 (Weaker than me), and student B perceived student A's skill as 2 (About the same), the absolute difference is $|1-2| = 1$. For the average of the scores, the value of pair-perceived skill is the average of 1 and 2 = 1.5.
- pair-compatibility ((C1–1) x (C2–1) for the pair, where C1 and C2 are the compatibility responses of each of the students in the pair (Very Compatible = 3, OK = 2, Not Compatible = 1). With this computation, pair-compatibility could have one of the following values: 0, 1, 2, or 4). The advantage of this computation is that when one partner perceives that the pair is Not Compatible, the value of pair-compatibility will be 0 since Not Compatible has a numerical code of 1.

3.4.1 Missing Value Problem

During the peer-evaluation survey, many students did not provide complete responses to the compatibility question. When the compatibility response was missing, the whole record was discarded. Additionally, when analyzing at the pair level, when the compatibility response from one partner was missing, the compatibility response from the other partner had to be discarded, even if available. Dropping all incomplete records caused a loss of valuable information. Values for other missing predictor variables were imputed. Imputing is used to predict a value for a missing value. Imputing the missing values using the SAS Enterprise Miner completed the records in which one or more of the predictor variables were missing.

The procedure used to impute the missing values with the SAS Enterprise Miner is described in Appendix B. At this point, the data set was completed with the missing values imputed. This completed data was analyzed with logistic regression with multivariate regression with the PROC GENMOD.

The original number of data points, number of data points after data cleaning and the percentage discarded are shown in Table 3.2.

Table 3.2: Data Points

Class	Original number of data points	Data points after data cleaning	% discarded
CS1	770	570	26%
SE	1258	796	36%
OO	104	56	54%

As observed in Table 3.2, the percentage of data points missing for the OO class is significantly higher than the CS1 and the SE classes because students in the CS1 and SE classes were penalized for not completing the peer evaluation.

Spearman's Rank Order (rho) Correlations were computed for the different factors including the Myers-Briggs personality type, SAT/GRE score, midterm, GPA, student reported subjective measure of self-esteem, gender, and ethnicity demographics, and partner-perceived skill level with compatibility as reported by the students. These correlations examine the hypotheses discussed in Section 3.

4. STUDENT RESEARCH RESULTS

In this chapter, the results of the student compatibility data analyses are explained. Section 4.1 outlines the results of the hypotheses of the research. Section 4.2 gives the results of the critical factors in pair compatibility as analyzed through a student survey. Section 4.3 explains the regression models for compatibility modeled on the multivariate analysis with experimental units as pair and individual, and further outlines the results of the hypotheses.

As shown in Table 4.1, only 6% of the students involved in the study reported incompatibility with their partner. Table 4.1 also provides student compatibility responses by gender. Generally, male students report more compatible experiences, though not a strong trend.

Table 4.1: Compatibility responses by gender

Class	N	Very Comp.	OK	Not Comp.
CS1- Female	116	60% (69)	34% (40)	6% (7)
CS1- Male	654	65% (422)	31% (206)	4% (26)
SE- Female	198	28% (56)	51% (101)	21% (41)
SE- Male	1060	68% (720)	29% (304)	3% (36)
OO- Female	18	72% (13)	28% (5)	0% (0)
OO- Male	86	77% (66)	13% (11)	10% (9)
Total	2132	63% (1346)	31% (667)	6% (119)

The distribution of compatibility responses by minority and non-minority (majority) students is reported in Table 4.2. There is a difference of 68 data points between the N in Tables 4.1 and 4.2 due to the missing ethnicity information for the Fall 2002 OO class. From the table, we observe that the compatibility of pairs including one or more minorities is comparable to that of majorities.

Table 4.2: Compatibility responses by ethnicity

Class	N	Very Compat.	OK	Not Compat.
CS1- Minority	90	62% (56)	35% (31)	3% (3)
CS1- Majority	680	64% (435)	32% (215)	4% (30)
SE- Minority	108	45% (49)	45% (49)	10% (10)
SE- Majority	1150	63% (727)	31% (356)	6% (67)
OO- Minority	3	100% (3)	0% (0)	0% (0)
OO- Majority	33	76% (25)	15% (5)	9% (3)
Total	2064	63% (1295)	32% (656)	5% (113)

4.1 Analysis of Correlations

In this section, the results of the Spearman's Rank Order correlation on the data are listed and explained. These correlations examine the hypotheses discussed in Section 3. Since the partner perceived skill (Better than me, About the same, Weaker than me), and compatibility (Very compatible, OK, Not compatible) are ordinal, and the actual skill (exam grade differences, SAT/GRE scores, GPA), and the programming self-esteem can be ranked, the Spearman Rank Order correlation was used in the analyses of the hypotheses. Spearman Correlation does not assume that the data are normally distributed. A p value of less than 0.05 indicates that a statistically significant correlation exists between the measure and compatibility.

4.1.1 Personality type and compatibility

*H1: Pairs are more compatible if students with different **Myers-Briggs personality type** are grouped together.*

Students reported their Myers Briggs personality type and their programming self-esteem at the beginning of the semester. The correlation between the Myers Briggs personality type of the students and their perceived compatibility was examined. The results, shown in Table

4.3, indicate that there exists no association between the personality type of the students and their partner compatibility. Therefore, we conclude that the personality type of the students is not a good predictor of their compatibility.

Table 4.3: Personality type and compatibility

Class	N	p-value	Correlation Coefficient
CS1	421	0.089	0.103
SE	903	0.827	-0.006
OO (fall 03)	30	0.901	-0.021

4.1.2 Actual skill and compatibility

H-2 Pairs are more compatible if students with similar actual skill level are grouped together.

In this section, we analyze the relationship between actual skill level and compatibility of students. Absolute differences in the partners' midterm grades, SAT, GRE, and overall GPA are used to assess multiple measures of actual skill level that might be correlated with compatibility. SAT/GRE scores and GPA were available only for the SE and the Fall 2003 OO classes since the Fall 2002 OO class did not sign the Informed Consent form. Gender and ethnicity information was also made available by the registrar data for these classes. Additionally, SAT score and ethnicity information was obtained for the CS1 class from the registrar data.

As indicated in Table 4.4, the greater the difference in the midterm of a CS1 female and her partner, the less likely she perceives compatibility with her partner. In SE class, the greater the difference in the midterm of partners, the less likely it is that they perceive compatibility. The greater the difference in the SAT of the SE students and the greater the difference in the GRE of the OO male students and their partner, the less likely it is that they

perceive compatibility with their partner. Similarly, the greater the difference in the GPA of the SE female student and her partner, the less likely it is that she perceives compatibility with her pairing partner. Students express a preference for working with a partner of similar skill level. These results indicate that it is not feasible for an instructor to proactively match students based on available measures of skill (midterm, SAT, GRE, GPA), except for the use of SAT in the SE class.

Table 4.4: Actual skill and compatibility

Measure	Class	Female	Male	Minority	Majority	All
		(*p*/correlation coefficient/N)				
Midterm	CS1	**0.012**/ -0.693/149	0.714/ -0.024/653	0.774/ -0.179/119	0.429/ -0.052/683	0.154/ -0.066/802
	SE	0.245/ -0.088/157	0.141/ 0.109/794	0.884/ 0.099/76	0.204/ -0.044/875	**0.000**/ -0.604/951
	OO	0.518/ 0.207/19	0.242/ 0.128/127	0.391/ 0.068/3	**0.002**/ 0.516/143	0.375/ -0.023/146
SAT	CS1	0.088/ -0.166/145	0.971/ -0.003/676	0.398/ -0.070/120	0.605/ -0.038/701	0.602/ -0.006/821
	SE	**0.013**/ -0.606/157	**0.002**/ -0.981/638	**0.000**/ -0.544/79	**0.027**/ -0.547/716	**0.000**/ -0.624/795
GRE	OO	0.287/ 0.335/7	**0.001**/ -0.620/14	0.394/ 0.077/3	0.197/ 0.446/18	0.246/ 0.172/21
GPA	SE	**0.019**/ -0.510/157	0.402/ 0.062/780	0.653/ -0.001/79	0.291/ 0.037/858	0.066/ 0.020/937
	OO	0.084/ -0.519/7	0.368/ 0.192/14	**0.025**/ -0.629/3	0.115/ -0.503/18	0.704/ 0.012/21

4.1.3 Partner-perceived skill and compatibility

H-3 Pairs are more compatible if students with similar perceived skill are grouped together.

In this analysis, a correlation between the partner-perceived skill level and the compatibility of the students was computed. Table 4.5 shows that partner-perceived skill level is strongly correlated with partner compatibility, suggesting that students in the computer science classroom perceive compatibility when they perceive that they have a

similar skill level with their partner. However, perceived skill level cannot be used as a proactive means of improving compatibility since perception, itself, cannot be predicted.

Table 4.5: Partner-perceived skill and compatibility

Class	Female	Male	Minority	Majority	All
	(*p*/correlation coefficient/N)				
CS1	**0.006/** 0.521/151	**0.019/** 0.579/620	**0.050/** 0.588/111	**0.001/** 0.795/660	**0.019/** 0.776/771
SE	**0.001/** 0.682/153	**0.000/** 0.671/771	**0.009/** 0.671/87	**0.000/** 0.813/837	**0.000/** 0.510/924
OO	**0.001/** 0.688/28	**0.000/** 0.528/76	**0.029/** 0.543/3	**0.000/** 0.699/101	**0.000/** 0.547/104

4.1.4 Programming self-esteem and compatibility

H-4 Pairs are more compatible if students with similar programming self-esteem are grouped together.

We wanted to see if there exists a correlation between the students' self-reported programming self-esteem and their partner compatibility. Therefore, in this analysis, the students' self-reported subjective measure of programming self-esteem was analyzed with their compatibility. Table 4.6 shows that in the SE class, the students' programming self-esteem has a correlation with their partner compatibility. This result suggests that the greater the difference in the programming self-esteem of the pair in SE, the more likely they are to perceive compatibility. The result is also true for the CS1 minority students. However, the greater the difference in the programming self-esteem of the OO female students and their partner's, the less likely they are to perceive compatibility. The result for the SE class does not agree with the earlier study done by researchers in Wales discussed in Section 2 (Thomas, Ratcliffe et al. 2003), which indicated that students with lower programming self-esteem liked pair programming more than students with higher programming self-esteem.

Table 4.6: Programming self-esteem and compatibility

Class	Female	Male	Minority	Majority	All
	(*p*/correlation coefficient/N)				
CS1	0.469/ 0.148/39	0.824/ 0.011/122	**0.003**/ 0.537/42	0.072/ 0.176/119	0.117/ 0.140/161
SE	**0.040**/ 0.588/63	**0.028**/ 0.514/348	**0.005**/ 0.590/73	**0.008**/ 0.879/338	**0.000**/ 0.501/411
OO	**0.009**/ -0.501/7	0.696/ -0.084/47	0.065/ -0.488/3	0.077/ -0.051/51	0.913/ -0.122/54

4.1.5 Work ethic and compatibility

H-7 Pairs are more compatible if students with similar work ethic are grouped together.

As will be discussed in Section 4.2, we surveyed CS1 and SE students about the most important factors in pair compatibility. The survey was completed in Fall 2003, after most of the data collected in this research was completed. One factor the students felt was important in pair compatibility that we had not examined was work ethic. Students with a strong work ethic might be striving for a high grade while students with a low work ethic might be more apathetic about their actual grade and might work less. As a result, at the start of the SE class in Fall 2004, we asked students to respond to the following question on a nine-point Likert scale:

In your class, do you work hard enough to:

1: just barely get by

9: get the best grade you possibly can

This data analysis computed a correlation between the differences in work ethic of pairs and their compatibility. Table 4.7 indicates that work ethic has no association with the compatibility of the students.

Table 4.7: Work ethic and compatibility

Class	N	*p*-value	Correlation Coefficient
SE (fall 2004)	136	0.161	0.121

4.1.6 Time management and compatibility

H-8 Pairs are more compatible if students with similar time management preferences are grouped together.

Another factor the students surveyed in Fall 2003 felt was important that we had not examined was time-management preferences. Some students prefer to start working on assignments very early, while some students prefer to start late. As a result, at the start of the SE class in Fall 2004, we asked students to respond to the following question on a nine-point Likert scale:

When you have homework, when do you get started?

1: very early

9: very late

Similar to work ethic, to find a correlation between the time management preferences of the students and their compatibility, this analysis was computed with the Fall 2004 SE class data. Table 4.8 indicates that there exists no correlation between the time-management preferences of students and their pair compatibility; however, the negative correlation coefficient suggests that the greater the difference in the time-management preferences of the pair, the less likely they are to perceive compatibility with their partner.

Table 4.8: Time management and compatibility

Class	N	*p*-value	Correlation Coefficient
SE (fall 2004)	136	0.686	-0.035

4.1.7 Partner-perceived skill and actual skill

We wanted to see how accurately students were able to guess their partner's actual skill level as demonstrated by midterm grades, SAT, GRE, and overall GPA. In this analysis, a correlation was computed for the partner-perceived and the actual skill of the students. Midterm grades, SAT, GRE, and overall GPA are used to assess multiple measures of actual skill that might be correlated with partner-perceived skill of the student. In Table 4.9, we observe that generally there exists no correlation between the partner-perceived skill and the actual skill of the student. However, in the CS1 class, female students with higher midterm grades are perceived weak (p = 0.002), and the higher the GPA of a minority student in the OO class, the more likely it is that the student is perceived academically strong.

Table 4.9: Partner-perceived skill and actual skill

Measure	Class	Female	Male	Minority	Majority	All
		(p/correlation coefficient/N)				
Midterm	CS1	**0.002**/ -0.748/149	0.478/ 0.055/653	0.506/ -0.122/119	0.114/ -0.121/683	0.058/ -0.143/802
	SE	0.297/ 0.143/157	0.192/ -0.097/767	0.642/ -0.055/76	0.862/ -0.066/875	0.701/ -0.015/924
	OO	0.630/ -0.122/19	0.395/ 0.171/85	0.065/ -0.222/3	0.076/ -0.313/143	0.928/ -0.009/104
SAT	CS1	0.416/ 0.157/145	0.885/ -0.013/626	0.393/ -0.162/120	0.684/ -0.035/651	0.816/ -0.020/771
	SE	0.220/ -0.165/157	0.667/ 0.032/638	0.648/ 0.057/79	0.360/ -0.033/716	0.063/ -0.080/795
GRE	OO	0.312/ -0.280/7	0.518/ -0.174/14	0.394/ 0.137/3	0.345/ 0.334/18	0.186/ -0.240/21
GPA	SE	0.489/ -0.057/149	0.453/ -0.055/775	0.143/ -0.180/79	0.333/ -0.034/845	0.062/ -0.077/924
	OO	0.324/ -0.264/7	0.571/ -0.135/14	**0.025**/ 0.634/3	0.908/ -0.040/18	0.227/ -0.206/21

4.1.8 Programming self-esteem and actual skill

We were interested in the relationship between students' self-perceived programming self-esteem and their demonstration of their programming skill as measured by midterm,

SAT, GRE, and the overall GPA. Therefore, we examined whether there was a correlation between the students' programming self-esteem and their actual skill. The results shown in Table 4.10 indicate that the higher the SAT of the CS1 female students, the more likely it is that they have a higher programming self-esteem. With the OO students, it is observed that the higher their GRE, and the higher the GPA of the male and the majority students, the lower they perceive their programming self-esteem. In general, however, there does not appear to be a correlation between the students' actual skill and programming self-esteem.

Table 4.10: Programming self-esteem and actual skill

Measure	**Class**	**Female**	**Male**	**Minority**	**Majority**	**All**
		(*p*/correlation coefficient/N)				
Midterm	CS1	0.055/ -0.405/39	0.322/ -0.097/122	0.459/ -0.162/42	0.328/ -0.095/119	0.147/ -0.127/161
	SE	0.469/ 0.167/63	0.936/ 0.011/348	0.390/ 0.045/73	0.213/ -0.105/338	0.301/ 0.047/411
	OO	0.801/ 0.074/7	0.923/ 0.025/47	0.065/ 0.132/3	0.856/ 0.026/51	0.873/ 0.030/54
SAT	CS1	**0.040**/ 0.587/39	0.429/ 0.089/122	**0.001**/ 0.657/42	0.592/ 0.060/119	0.074/ 0.179/161
	SE	0.529/ 0.154/63	0.177/ -0.160/348	0.361/ -0.159/73	0.163/ -0.076/338	0.503/ -0.029/411
GRE	OO	**0.030**/ -0.611/7	**0.015**/ -0.708/14	**0.025**/ -0.583/3	**0.050**/ -0.667/18	**0.017**/ -0.556/21
GPA	SE	0.474/ 0.150/63	0.546/ 0.068/348	0.529/ -0.112/79	0.926/ -0.005/332	0.823/ 0.009/411
	OO	0.328/ -0.261/7	**0.028**/ -0.560/14	0.065/ 0.133/3	**0.028**/ 0.503/18	0.910/ -0.020/21

Figure 4.1 depicts the negative association between the GRE and the programming self-esteem of the graduate students. In the figure, we observe that the higher the GRE of the OO students, the lower they perceive their programming self-esteem.

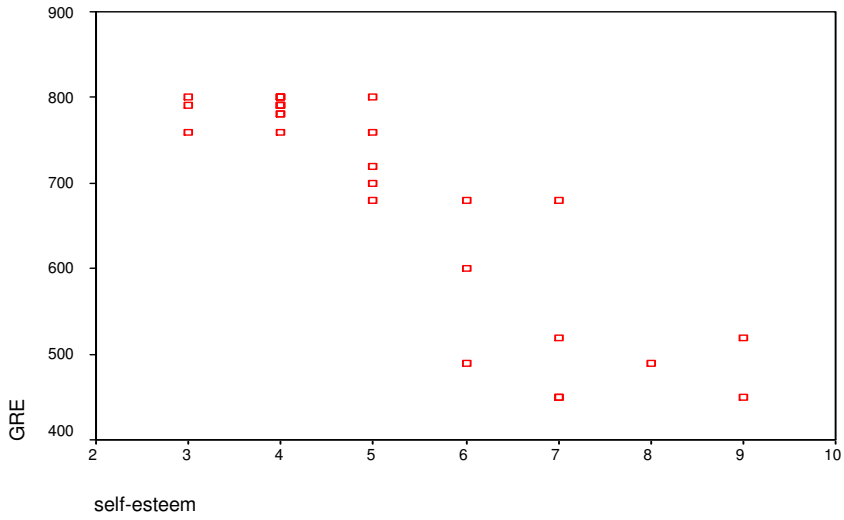

Figure 4.1: Programming self-esteem and actual skill of OO students

4.2 Factors in pair compatibility – a student survey

The CS1, SE, and OO students in Fall 2003 were surveyed about the most important factors in pair compatibility. The survey was filled by 110 students after most of the data collection in this research was completed. Completion of the survey was handled via a web application and was not mandatory for the students. Table 4.11 shows the results of the question: "Which qualities are most important in a compatible partner?" The two most important factors are having a partner with a similar work ethic and grade ambition. Because of these indications, we analyzed the relationship between these factors and compatibility. However, our results did not support the importance of these factors to compatibility.

Table 4.11: Qualities of compatible partners

Quality	CS1 (N=34)	SE (N=69)	OO (N=7)	Total
Work ethic	21	50	6	77
Grade ambition	17	48	4	69
Similar skill	21	42	5	68
Just comfortable	17	41	5	63
Personality match	17	42	5	60
Sense of humor	18	38	4	60
Participation	17	37	4	58
Project mgmt skills	13	38	2	53
Punctual for meetings	11	38	4	53
Work pattern	9	37	4	50
Same gender	15	30	3	48
Similar age	13	17	2	32
Higher skill	9	13	2	24
Diff gender	8	9	1	18
Same nationality	6	7	0	13
Same ethnicity	5	7	0	12
Lower skill	2	2	0	4

The students were also asked, “Which qualities made you incompatible with a partner?” Table 4.12 shows the results of this question on the survey. The two important factors that can negatively affect compatibility were having a partner with a personality mismatch and having a partner with a lower skill. Again, our correlation analysis of these two factors and compatibility did not support the results of the survey.

Table 4.12: Qualities of incompatible partners

Quality	CS1 (N=34)	SE (N=69)	OO (N=7)	Total
Personality mismatch	17	30	3	50
Lower skill	14	31	4	49
Diff work ethic	14	29	3	46
Diff work patterns	5	34	3	42
Diff project mgmt skills	0	29	2	31
Not participate	7	20	1	28
Same gender	5	15	2	22
Diff grade ambition	11	10	0	21
Doesn't talk enough	4	14	1	19
No sense of humor	5	14	0	19
Too talkative	5	13	1	19
Diff gender	9	7	1	17
Body odor	3	12	0	15
Breath problems	3	11	0	14
Age difference	4	6	0	10
Makes feel inferior	0	10	0	10
Not punctual	1	8	1	10
Diff ethnicity	1	8	0	9
Language problems	4	4	1	9
Higher skill	0	7	0	7
Similar skill	2	5	0	7
Asked personal questions	1	3	0	4

4.3 Compatibility Analysis

In this section, the multivariate analysis of the predictor variables with compatibility is discussed. Multivariate analysis enables one to look at multiple predictor variables and find an association to identify the principal factors that contribute to the compatibility responses. These results will be used to support the results obtained in Section 4.1 and to examine the

hypotheses in Section 3. Pair-level analysis results are outlined in Section 4.3.1, and Section 4.3.2 gives the result of the individual level analysis. We computed the values using both the individual and the pair as an experimental unit to find any association between the individual students and their compatibility response, and between pairs and their compatibility response.

4.3.1 Pair-level compatibility analysis

Below are the pair-level results of the multivariate analysis, analyzing the factors influencing the compatibility of a pair. In Table 4.13, the greater the difference in the perceived skill between the pair, the less likely the pair perceives compatibility. Pairs with different gender are less likely to be compatible in the undergraduate classes. These results are consistent with Section 4.1 results. Female-female pairs are more likely to report compatibility, supporting hypothesis H-5 (*Pairs are more compatible if students with same gender are grouped together*), though not statistically significant. The gender results are true for the undergraduate classes as seen in the table. The SAS output is shown in Appendix A. No correlation is significant with the OO class fall 2003 as seen in Figure A.4 in Appendix A.

Table 4.13: Pair-level compatibility analysis

Factor	**CS1**	**SE**	**OO** ('02)	**OO** ('03)
	(*p*/correlation coefficient/N)			
Pair-Perceived Skill	**0.019**/-0.383/285	**0.001**/-0.699/398	0.680/0.216/16	-
Pair-midterm	0.873/-0.007/285	0.124/0.012/398	0.832/-0.008/16	-
Pair-SAT/Pair-GRE	0.186/-0.010/285	0.389/-0.006/398	-	-
Pair-GPA	0.416/-0.016/285	0.639/-0.099/398	-	-
Pair-Gender (FF)	0.483/0.406/285	0.842/0.099/398	-	-
Pair-Gender (MF)	**0.015**/-0.665/285	**0.013**/-0.535/398	-	-
Pair-Gender (MM)	-	-	-	-
Pair-Ethnicity(MM)	0.919/0.104/285	0.256/0.291/398	-	-
Pair-Ethnicity(NM)	0.415/0.245/285	0.277/0.268/398	-	-
Pair-Ethnicity(NN)	-	-	-	-

OO data is analyzed separately by the semesters because GRE/GPA/ethnicity data were available only for Fall 2003 OO class.

4.3.2 Individual-level compatibility analysis

With an *individual* as the experimental unit, the factors influencing the compatibility of different individuals are analyzed. The approach to this analysis differs from Section 4.1 in that we look at multiple predictor variables influencing the compatibility of an individual. From Table 4.14, we observe that when students perceive their partner to be weaker than themselves, they are less likely to perceive compatibility. Students with higher SAT scores in CS1 are less likely to report compatibility with their partner. Female students are less likely to report compatibility than male students in the undergraduate classes. Though this result is not statistically significant in CS1, the negative estimate coefficient suggests the result. On the contrary, graduate female students are likely to report compatibility (not statistically significant). Students with higher midterm grades are less likely to report compatibility in the SE class. These results are consistent with the results in Section 4.1. Minority students in OO are more likely to report compatibility, and so are the graduate male students. The SAS output is shown in Appendix A.

Table 4.14: Individual-level compatibility analysis

Factor	**CS1**	**SE**	**OO** (‘02)	**OO** (‘03)
	(*p*/correlation coefficient/N)			
Perceived Skill	**0.001**/-1.358/570	**0.001**/-3.693/796	**0.004**/-2.492/32	**0.001**/-77.576/14
Midterm	0.840/-0.006/570	**0.009**/-0.023/796	0.214/-0.022/32	–
SAT/GRE	**0.008**/-0.006/570	0.788/-0.002/796	–	–
GPA	0.532/0.069/570	0.630/0.093/796	–	–
Gender (F)	0.155/-0.299/570	**0.009**/-0.619/796	0.999/23.761/32	–
Gender (M)	–	–	–	**0.001**/53.500/14
Ethnicity (M)	0.878/-0.036/570	0.213/-0.388/796	–	**0.001**/17.460/14
Ethnicity (N)	–	–	–	–

A hyphen "–" in the table indicates that no association at all was reported for the correlation.

5. INDUSTRY SURVEY

This chapter outlines the description and the results of a survey that was used to study the factors influencing the compatibility of pair programmers working in industry. The survey, similar to the one given to the students, can be found in Appendix C. The survey was validated by nine students and two professors in the NCSU Software Engineering group before being posted on the message boards.

The survey primarily consisted of closed-ended questions because such questions provide the same frame of reference for all participants when evaluating the responses, allowing for quantitative analysis of the responses, and it is also easy and inexpensive to analyze the data. For questions where it was difficult to anticipate all possible choices, participants were given the option to specify their own answer if it was not available in the list of choices provided. Participants could respond in an 'Other' text box. The open-ended 'Other' text boxes were designed to elicit subjective responses.

An invitation to take the survey was posted on multiple message boards and online-forums on the internet (one MSN message board, two Google message boards, and 43 Yahoo Groups). The web-based survey allowed for geographically diverse population to be sampled relatively quickly and at a low cost.

5.1 Survey Results

A total of 339 responses were obtained for the survey. This section gives the percentage of responses for each of the questions in tables and graphs.

5.1.1 Qualities of most compatible partners

Question: "Think about all the partners you have ever pair programmed with. Consider which of these you have been the most compatible with. What qualities made you compatible with those partners? Check ALL that apply."

Table 5.1 gives the percentage of responses by the respondents to the options provided. As observed in the table, the work ethic, sense of humor, personality, and skill of the pairing partner are of highest priority of the respondents in seeking their compatible partner.

Table 5.1: Qualities of most compatible partners

Qualities	Responses
Work ethic	71% (240)
Sense of humor	65% (221)
Personality match	61% (207)
Similar skill level	61% (206)
Felt comfortable	56% (191)
Work patterns	44% (150)
Work participator	40% (135)
Higher skill level	33% (111)
Same gender	27% (92)
Project mgmt skills	25% (84)
Punctual	22% (76)
Different gender	21% (73)
Lower skill level	20% (69)
Similar age	15% (51)
Same ethnicity	7% (25)
Same nationality	6% (21)

5.1.2 Benefits of Working with a Compatible Partner

Question: "In what ways is it beneficial to have a compatible partner? Check ALL that apply:"

Table 5.2 lists the percentage of responses to the options provided for the question. From the table, the most important benefit of working with a compatible partner is it makes the programming experience more enjoyable.

Table 5.2: Benefits of working with compatible partners

Benefits	Percentage Responses
Enjoyable	88% (293)
More learning	85% (283)
More productive	82% (272)
Higher quality code	75% (250)

5.1.3 Qualities of Least Compatible Partners

Question: "Think about all the partners you have ever pair programmed with. Consider which of these you have been the least compatible with. What qualities made you incompatible with those partners? Check ALL that apply."

Table 5.3 outlines the percentage of responses to the options provided. From the results, we observe that not having a personality match with the pairing partner affected a majority of the respondents on the survey. Responses to this question suggest that differences in ethnicity and nationality does not critically affect the compatibility of the survey respondents.

Table 5.3: Qualities of least compatible partners

Qualities	% Responses
Personality mismatch	55% (182)
Differences in work ethic	44% (146)
Not participatory	44% (145)
Lower skill level	32% (108)
Sense of humor	28% (93)
Did not talk enough	27% (91)
Breath problems	20% (65)
Body odor	19% (64)
Different work patterns	18% (61)
Different PM skills	17% (58)
Too talkative	17% (57)
Made you inferior	16% (52)
Language problems	15% (50)
Not punctual	13% (42)
Ask personal questions	7% (22)
Higher skill level	6% (21)
Different gender	5% (17)
Same gender	4% (13)
Similar skill level	4% (13)
Age differences	3% (9)
Different nationality	2% (8)
Different ethnic background	2% (6)

5.1.4 Setbacks of Working with an Incompatible Partner

Question: "What setbacks do you face when you are incompatible with your partner? Check ALL that apply."

Table 5.4 lists the percentage of responses to the options provided for the question. The results indicate that pairing with an incompatible partner makes the programming experience less enjoyable to a majority of the respondents.

Table 5.4: Setbacks of working with incompatible partners

Setbacks	% responses
Less enjoyable	79% (263)
Less productive	77% (257)
Miss learning opportunities	56% (185)
Low quality code	39% (131)

5.1.5 Experience in Pair Programming

Question: "How long have you been pair programming?"

The responses to the options provided are observed in Table 5.5.

Table 5.5: Experience pair programming

Experience pair programming	% responses
More than 2 years	44% (147)
Between 6 months and 1 year	20% (67)
Less than 6 months	18% (61)
Between 1 and 2 years	17% (58)

5.1.6 Percentage of Day Pairing

Question: "On average, what percentage of the day do you pair program?"

The responses to the options provided are observed in Table 5.6. We observe from the table that the responses to the survey were obtained from practitioners pairing in a range of percentage throughout the day on an average.

Table 5.6: Percentage of day pairing

Percentage of day pairing	% responses
Less than 10%	28% (94)
Between 10% and 25%	23% (75)
Between 25% and 50%	17% (58)
More than 75%	16% (55)
Between 50% and 75%	15% (51)

5.1.7 Enjoyment of pair programming

Question: "Do you enjoy pair programming?: Yes/No. If yes, why? If no, why?"

The responses to the answer provided on the survey are listed in Table 5.7.

Table 5.7: Enjoyment of pair programming

Enjoy pair programming?	Reason	
Yes 91% (304)	Learn	77% (257)
	Higher quality code	73% (244)
	More productive	71% (236)
	Social aspect	57% (190)
	Avoid long debugging sessions	49% (156)
No 9% (29)	Like working alone	5% (16)
	Get more work done alone	5% (15)
	Feel like I'm teaching my partner all the time	4% (14)
	Produce high quality code on my own	3% (11)
	No sense of accomplishment	2% (8)

5.1.8 Country of Origin

Question: "What is your country of origin?"

As observed in Table 5.8, the survey respondents were from 38 different countries, improving the validity of the geographic distribution of the survey.

Table 5.8: Country of origin

Country	# respondents
USA	158
India	47
Canada	22
Israel	13
UK	10
Australia	9
Germany	7
Brazil	6
China	6
Indonesia	4
Sweden	4
Turkey	4
Bulgaria	3
Netherlands	3
New Zealand	3
South Africa	3
Belgium	2
Egypt	2
England	2
Mexico	2
Norway	2
Philippines	2
Poland	2
Portugal	2
Russia	2

Responses were also obtained from other thirteen countries.

5.1.9 Country of Work

Additionally, the respondents were asked to state their country of work in the survey.

The results to the question are same as the results in Table 5.8 in the survey question above.

5.2 Responses to open-ended options

For each question, the categories that were coded for the responses are provided, and then the open-ended responses are listed under the respective categories. Additionally,

the responses that were not categorized are listed under the “uncategorized” category. The open-ended responses are listed in Appendix D.

5.3 Responses posted on message boards

The survey results, when posted on the message boards, generated many responses via message board posts. These open-ended responses are attached in Appendix E.

6. CONCLUSIONS

We analyzed the pair compatibility of students in the computer science classroom by analyzing the factors that we felt had the potential to influence their compatibility. The factors we looked at are: Myers Briggs personality type, actual skill (measured by the multiple measures of midterm grade, overall GPA, and SAT/GRE scores), perceived skill by the partner, student-reported subjective measure of self-esteem, gender and ethnicity demographics, work ethic, and time management preferences of the students.

Table 6.1 provides a summary of the results of for this research with the student data.

Table 6.1: Results Summary

	Hypothesis Pair are more compatible if students with …	**CS1**	**SE**	**OO**
H-1	… different **personality type** are grouped together	No	No	No
H-2	… similar **actual skill** level are grouped together	No	Yes	No
H-3	… similar **perceived skill** are grouped together	Yes	Yes	Yes
H-4	… similar **self-esteem** are grouped together	No	Yes	No
H-5	… same **gender** are grouped together	No	No	No
H-6	… similar **ethnicity** are grouped together	No	No	No
H-7	… similar **work ethic** are grouped together		No	
H-8	… similar **time management** are grouped together		No	

The most prominent result of the student data is the weaker a student perceives his or her partner to be, the less likely the student is compatible with that partner. However, educators cannot be proactive about matching students in pairs based on perceived skill because there were no measures available to the educator (GPA, SAT, midterm, GRE) that were found to be consistent predictors of a student's perception of the skill level of their partner. Mixed gender pairs are less likely to perceive compatibility. Additionally, pairing of two female students in a pair is likely to result in a compatible pair in the undergraduate classroom.

The personality type of the students is not a significant factor in predicting the compatibility of the students. The greater the difference in the programming self-esteem of the undergraduate software engineering students, the more likely they are to be compatible, while the greater the difference in the programming self-esteem of the graduate female students and their partner, the less likely it is that they perceive compatibility. Female students with higher midterm grades are perceived as weak by their partners in CS1.

OO male students with greater differences in the GRE score with their partner are less likely to perceive compatibility. The greater the difference in the overall GPA of a female student in the SE class and her partner, the less likely they are to perceive compatibility.

The higher the GRE of the graduate students, the lower is the student's programming self-esteem, while the lower the GPA of the graduate male students, the higher their programming self-esteem. The higher the GPA of the non-minority students in the OO class, the higher they perceive their programming self-esteem.

As the difference in the SAT of the pair increases in the SE class, the less likely they are to perceive compatibility. The greater the difference in the overall GPA of a minority student in the OO class and his or her partner, the less likely they are to perceive compatibility.

Additionally, work ethic and the time-management priorities of the students are not a factor in predicting their pair compatibility.

Results of the industry survey of pairing partners revealed that respondents feel more compatible with a partner who has a similar work ethic. A few additional qualities of the compatible partners were: self-motivated, effective communicator, had complementary skills, similar skill level, different project-management skills, similar work drive, and adaptability.

On the industry survey, 88% respondents believed that a significant benefit of working with a compatible partner is that the programming experience is more enjoyable. Some other benefits suggested by the respondents include time management, good communication, less stress, and knowledge sharing.

The respondents indicated the following qualities for their least compatible partners: complaining, lacking the right background, not flexible, inability to communicate, and difficult to motivate. A major setback of pairing with an incompatible partner is that it makes the programming experience less enjoyable, as indicated by 79% of the respondents.

Respondents who enjoyed pair programming attribute their enjoyment to the learning aspect of the paradigm. The survey also revealed that 16 people preferred solo programming because they feel that they get more work done alone.

Results from this research can help alleviate software engineering instructors' organizational concerns in pairing of students by increasing the likelihood of proactively forming compatible pairs. Additionally, instructors can help improve the programming experience of female and minority students in hopes of increasing their retention rates in the information technology field. The results from the responses from the industry can likely be used to form mutually compatible pairs in the industry.

Future research related to this thesis should focus on examining the factors in the compatibility of industry pair programmers in detail. Through such a detailed evaluation, factors of the compatibility of pair programmers could potentially be identified, and the chances of forming mutually-compatible pairs can be improved.

LIST OF REFERENCES

Beck, K. (2000). Extreme programming explained: embrace change, Addison-Wesley.

Bevan, J., L. Werner, et al. (2002). Guidelines for the User of Pair Programming in a Freshman Programming Class. Conference on Software Engineering Education and Training, Kentucky.

Beyer, S., K. Rynes, et al. (2003). Gender Differences in Computer Science Students. ACM SIGCSE, Reno, Nevada.

Bishop-Clark, C. and D. Wheeler (1994). "The Myers-Briggs personality type and its relationship to computer programming." Journal of Research on Computing in Education **26**: 358-370.

Cockburn, A. and L. Williams (2001). The Costs and Benefits of Pair Programming. Extreme Programming Examined, Addison-Wesley.

Cohoon, J. M. (May 2003). Must There Be So Few? Including Women in CS. International Conference on Software Engineering, Portland, Oregon.

Doolittle, P. E. (1997). "Vygotsky's Zone of Proximal Development as a Theoretical Foundation for Cooperative Learning." Journal on Excellence in College Teaching **8**: 83-103.

Hornig, L. S. (1984). "Women in science and engineering: Why so few?" Tech. Rev. **87,8**: 31-41.

Jung, C. G. (1971). Psychological Types. Princeton U.P.

Katira, N., L. Williams, et al. (2004). On Understanding the Compatibility of Student Pair Programmers. ACM SIGCSE, Norfolk, VA.

Keirsey, D. (1998). Please Understand Me II. Del Mar, CA, Prometheus Nemesis Book Company.

Margolis, J. and A. Fisher (1997). Geek Mythology and Attracting Undergraduate Women to Computer Science. Joint National Conference in Engineering Program Advocates Network and the National Association of Minority Engineering Program Administrators.

McDowell, C., L. Werner, et al. (2002). The Effect of Pair Programming on Performance in an Introductory Programming Course. ACM Special Interest Group of Computer Science Educators, Kentucky.

McDowell, C., L. Werner, et al. (2003). The Impact of Pair Programming on Student Performance of Computer Science Related Majors. International Conference on Software Engineering 2003, Portland, Oregon.

Mussweiler, T. and G. V. Bodenhausen (2002). "I know you are, but what am I? Self-evaluative consequences of judging ingroup and outgroup members." Journal of Personality and Social Psychology **82**: 19-32.

Nagappan, N., L. Williams, et al. (2003). Improving the CS1 Experience with Pair Programming. SIGCSE Technical Symposium on Computer Science Education.

Nagappan, N., L. Williams, et al. (2003). Improving the CS1 Experience with Pair Programming. SIGCSE 2003.

Nagappan, N., L. Williams, et al. (2003). Pair Learning: With an Eye Toward Future Success. Extreme Programming/Agile Universe, New Orleans, Springer-Verlag Lecture Notes in Computer Science.

Nelson, C. (1996). "Student Diversity Requires Different Approaches to College Teaching, Even in Math and Science." American Behavioral Scientist **40**: 165-174.

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

Palmieri, D. (2002). Knowledge Management Through Pair Programming. Computer Science. Raleigh, NC. North Carolina State University. Master of Science. **88**.

Rayman, P. and B. Brett (July/August 1995). "Women Science Majors: What Makes a Difference in Persistence after Graduation?" The Journal of Higher Education.

Shaw, J. I. and W. N. Steers (2001). "Gathering Information to Form an Impression: Attribute Categories and Information Valence." Current Research in Social Psychology **6(1)**: 1-17.

Statistics, N. C. f. E. (2002). Digest of Education Statistics. U. S. D. o. E. Institute of Education Sciences.

Thomas, L., M. Ratcliffe, et al. (2003). Code Warriors and Code-a-Phobes: A study in attitude and pair programming. *Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education*.

Treisman, U. (1992). "Studying students studying calculus: A look at the lives of minority mathematics students in college." College Mathematics Journal **23**: 362-372.

Vogel, G. (2004). Behavioral Evolution: The Evolution of the Golden Rule. Science Magazine. **303:** 1128-1131.

Vygotsky, L. (1978). Mind in Society: The Development of Higher Psychological Processes. Cambridge, MA, Harvard University Press.

Weber-Wulff, D. (2000). Combating the Code Warrior: A Different Sort of Programming Instruction. ITiCSE 2000, Helsinki, Finland, ACM Press.

Williams, L. (2000). The Collaborative Software Process. Department of Computer Science. Salt Lake City, UT.

Williams, L. and R. Kessler (2003). Pair Programming Illuminated. Reading, Massachusetts, Addison Wesley.

Williams, L., R. Kessler, et al. (2000). "Strengthening the Case for Pair Programming." IEEE Software **17**(4): 19-25.

Williams, L., C. McDowell, et al. (2003). Building Pair Programming Knowledge Through a Family of Experiments. International Symposium on Empirical Software Engineering (ISESE) 2003, Rome, Italy.

Appendix A: SAS Output

Analysis Of GEE Parameter Estimates
Empirical Standard Error Estimates

Parameter		Estimate	Standard Error	95% Confidence Limits		Z	Pr > |Z|
Intercept1		0.6033	0.2588	0.0961	1.1106	2.33	0.0197
Intercept2		2.0346	0.2836	1.4787	2.5905	7.17	<.0001
Intercept3		3.7660	0.3823	3.0166	4.5154	9.85	<.0001
pper		-0.3831	0.1633	-0.7030	-0.0631	-2.35	0.0190
pmidterm		-0.0007	0.0042	-0.0088	0.0075	-0.16	0.8736
pgpa		-0.0168	0.0229	-0.0617	0.0280	-0.74	0.4615
pSAT		-0.0103	0.0078	-0.0256	0.0050	-1.32	0.1866
pMBTI		0.1643	0.2884	-0.4009	0.7295	0.57	0.5689
pGender	FF	0.4068	0.5803	-0.7306	1.5441	0.70	0.4833
pGender	MF	-0.6654	0.2739	-1.2023	-0.1285	-2.43	0.0151
pGender	MM	0.0000	0.0000	0.0000	0.0000	.	.
pethnicity	MM	0.1048	1.0362	-1.9261	2.1357	0.10	0.9195
pethnicity	NM	0.2459	0.3019	-0.3458	0.8376	0.81	0.4153
pethnicity	NN	0.0000	0.0000	0.0000	0.0000	.	.

Figure A.1: Multivariate Analyses CS1 – pair level

Analysis Of GEE Parameter Estimates
Empirical Standard Error Estimates

Parameter		Estimate	Standard Error	95% Confidence Limits		Z	Pr > |Z|
Intercept1		-1.2869	0.3435	-1.9602	-0.6136	-3.75	0.0002
Intercept2		-0.4849	0.3364	-1.1443	0.1745	-1.44	0.1495
Intercept3		0.6307	0.3390	-0.0337	1.2951	1.86	0.0628
pper		0.6997	0.1431	0.4192	0.9801	4.89	<.0001
pmidterm		0.0124	0.0081	-0.0034	0.0282	1.54	0.1245
pGPA		-0.0999	0.2133	-0.5181	0.3182	-0.47	0.6395
pSAT		-0.0006	0.0007	-0.0020	0.0008	-0.86	0.3895
pmyers		-0.1379	0.2929	-0.7120	0.4362	-0.47	0.6379
pgender	FF	-0.0994	0.4985	-1.0764	0.8776	-0.20	0.8420
pgender	MF	-0.5350	0.2165	-0.9592	-0.1107	-2.47	0.0135
pgender	MM	0.0000	0.0000	0.0000	0.0000	.	.
pethnicity	MM	0.2919	0.2571	-0.2120	0.7958	1.14	0.2563
pethnicity	NM	0.2688	0.2473	-0.2158	0.7534	1.09	0.2770
pethnicity	NN	0.0000	0.0000	0.0000	0.0000	.	.

Figure A.2: Multivariate Analyses SE – pair level

Analysis Of GEE Parameter Estimates
Empirical Standard Error Estimates

Parameter	Estimate	Standard Error	95% Confidence Limits		Z	Pr > |Z|
Intercept1	0.8457	0.7919	-0.7065	2.3978	1.07	0.2856
Intercept2	2.0748	1.2135	-0.3037	4.4533	1.71	0.0873
Intercept3	2.8252	1.7166	-0.5394	6.1898	1.65	0.0998
pper	0.2167	0.5256	-0.8134	1.2467	0.41	0.6802
pfinal	-0.0083	0.0395	-0.0857	0.0690	-0.21	0.8328

Figure A.3: Multivariate Analyses OO fall 2002 – pair level

Analysis Of GEE Parameter Estimates
Empirical Standard Error Estimates

Parameter		Estimate	Standard Error	95% Confidence Limits		Z	Pr > |Z|
Intercept1		44.4582	1.7321	41.0634	47.8529	25.67	<.0001
Pmyers		-0.0000	3.5610	-6.9795	6.9795	-0.00	1.0000
pper		-88.8201	0.0000	-88.8201	-88.8201	.	.
pgpa		-0.0000	2.3970	-4.6980	4.6980	-0.00	1.0000
pfinal		0.0000	0.5599	-1.0973	1.0973	0.00	1.0000
pgender	MF	0.0000	0.0000	0.0000	0.0000	.	.
pgender	MM	0.0000	0.0000	0.0000	0.0000	.	.
pethnicity	NM	0.0000	0.0000	0.0000	0.0000	.	.
pethnicity	NN	0.0000	0.0000	0.0000	0.0000	.	.
Pself_esttem		0.0000	0.0000	0.0000	0.0000	.	.

Figure A.4: Multivariate Analyses OO fall 2002 – pair level

Analysis Of Initial Parameter Estimates

Parameter		DF	Estimate	Standard Error	Wald 95% Confidence Limits		Chi-Square	Pr > ChiSq
Intercept1		1	1.5629	0.4617	0.6581	2.4678	11.46	0.0007
Intercept2		1	4.2260	0.4950	3.2558	5.1962	72.89	<.0001
prcdskill	1	1	-1.3581	0.2215	-1.7922	-0.9240	37.60	<.0001
prcdskill	2	1	-0.2119	0.1777	-0.5602	0.1365	1.42	0.2332
prcdskill	3	0	0.0000	0.0000	0.0000	0.0000	.	.
SAT		1	-0.0067	0.0025	-0.0117	-0.0017	6.93	0.0085
GPA		1	0.0694	0.1113	-0.1487	0.2876	0.39	0.5327
gender	f	1	-0.2992	0.2105	-0.7119	0.1134	2.02	0.1552
gender	m	0	0.0000	0.0000	0.0000	0.0000	.	.
ethnicity	M	1	-0.0367	0.2397	-0.5065	0.4330	0.02	0.8782
ethnicity	N	0	0.0000	0.0000	0.0000	0.0000	.	.
midterm		1	-0.0006	0.0029	-0.0062	0.0051	0.04	0.8406
Scale		0	1.0000	0.0000	1.0000	1.0000		

Figure A.5: Multivariate Analyses CS1 – individual level

Analysis Of GEE Parameter Estimates
Empirical Standard Error Estimates

Parameter		Estimate	Standard Error	95% Confidence Limits		Z	Pr > |Z|
Intercept1		3.9443	1.0360	1.9138	5.9747	3.81	0.0001
Intercept2		6.2830	1.0527	4.2197	8.3463	5.97	<.0001
prcdskill	1	-3.6934	0.3377	-4.3553	-3.0315	-10.94	<.0001
prcdskill	2	-2.3457	0.3146	-2.9623	-1.7292	-7.46	<.0001
prcdskill	3	0.0000	0.0000	0.0000	0.0000	.	.
SAT		-0.0002	0.0007	-0.0015	0.0011	-0.27	0.7889
GPA		0.0938	0.1952	-0.2888	0.4765	0.48	0.6308
gender	f	-0.6194	0.2371	-1.0841	-0.1547	-2.61	0.0090
gender	m	0.0000	0.0000	0.0000	0.0000	.	.
ethnicity	1	-0.3889	0.3124	-1.0012	0.2234	-1.24	0.2132
ethnicity	2	0.0000	0.0000	0.0000	0.0000	.	.
midterm		-0.0234	0.0090	-0.0410	-0.0058	-2.60	0.0093

Figure A.6: Multivariate Analyses SE – individual level

Analysis Of Initial Parameter Estimates

Parameter		DF	Estimate	Standard Error	Wald 95% Confidence Limits		Chi-Square	Pr > ChiSq
Intercept1		1	3.0911	1.4489	0.2514	5.9308	4.55	0.0329
Intercept2		1	4.8196	1.5849	1.7132	7.9260	9.25	0.0024
prcdskill	1	1	-2.4920	0.8682	-4.1937	-0.7903	8.24	0.0041
prcdskill	2	1	0.4033	0.8207	-1.2052	2.0119	0.24	0.6231
prcdskill	3	0	0.0000	0.0000	0.0000	0.0000	.	.
gender	f	1	23.7611	141430.9	-277176	277223.2	0.00	0.9999
gender	m	0	0.0000	0.0000	0.0000	0.0000	.	.
final		1	-0.0229	0.0184	-0.0590	0.0132	1.54	0.2143
Scale		0	1.0000	0.0000	1.0000	1.0000		

Figure A.7: Multivariate Analyses OO (fall 2002) – individual level

Analysis Of GEE Parameter Estimates
Empirical Standard Error Estimates

Parameter		Estimate	Standard Error	95% Confidence Limits		Z	Pr > |Z|
Intercept1		-624.097	0.0000	-624.097	-624.097	-2.79E9	<.0001
Intercept2		-523.021	2.6448	-528.205	-517.837	-197.75	<.0001
prcdskill	1	-77.5769	2.0000	-81.4968	-73.6570	-38.79	<.0001
prcdskill	2	-26.6462	0.0000	-26.6462	-26.6462	.	.
prcdskill	3	0.0000	0.0000	0.0000	0.0000	.	.
gpa		78.4192	0.0000	78.4192	78.4192	.	.
gender	M	53.5002	0.6368	52.2522	54.7483	84.02	<.0001
gender	f	0.0000	0.0000	0.0000	0.0000	.	.
ethnicity	M	17.4600	1.1855	15.1365	19.7836	14.73	<.0001
ethnicity	N	0.0000	0.0000	0.0000	0.0000	.	.
mbti	ENFJ	-17.5542	1.0000	-19.5142	-15.5943	-17.55	<.0001
mbti	ENFP	-73.1365	1.1855	-75.4601	-70.8130	-61.69	<.0001
mbti	ESFJ	-68.7843	0.8600	-70.4699	-67.0988	-79.98	<.0001
mbti	ESFP	46.7349	2.3249	42.1781	51.2917	20.10	<.0001
mbti	ESTP	0.0000	0.0000	0.0000	0.0000	.	.
mbti	INTJ	0.0000	0.0000	0.0000	0.0000	.	.
selfesteem		-0.8642	0.0000	-0.8642	-0.8642	.	.
final		4.3564	0.0000	4.3564	4.3564	.	.

Figure A.8: Multivariate Analyses OO (fall 2003) – individual level

Appendix B: Imputing Missing Values

Start SAS system from the Start Menu (in Windows).

In SAS, select File-> import data -> (check Standard data source) -> select Microsoft Excel in the drop-down -> click Next -> specify the location of the spreadsheet by selecting the Browse button and click Next.

Then specify the name of the Member file in the Work library that will be used for the file and click Finish.

The procedure above selects the Excel data file to be used for the data manipulation in SAS.

Select the Enterprise Miner option in SAS with the following procedure.
Solutions -> Analysis -> Enterprise Miner

1. Create a new diagram
2. Add an Input Data Source Node and specify the source data as the Member file name in the Work library.
   Click the Variables tab and set the model role for Comp as "target".
   [read data sources and define their attributes, set the model role of each variable, display summary statistics for interval and class variables, and define target profile for the target in the data set.]
3. Right click on the Input Data Source Node and click Run.
4. Right click on the diagram to add a node - Tree node.
5. Draw an arrow from the Input Data Source node to the Tree node to create an association of the input data.
6. Right click on the Tree Node and click Run.
7. Right click on the diagram to add a node - Replacement node.
8. Draw an arrow from the Input Data Source node to the Replacement node to create an association of the input data.
9. In the Replacement node, under the tab 'Tree Imputation', select the tab 'Imputation Methods' and select "mean" for Interval variables and "most frequent value (count)" for Class variables.
   The Replacement node is used for imputing the missing values in the data set.
10. Right click on the Replacement Node and click Run.
    At this point, the data set was completed with the missing values imputed.

Appendix C: Pair Programming Compatibility Survey

This survey will be used to examine the factors that make pair programmers more or less compatible with each other. It will be used for software engineering research done at the North Carolina State University under the direction of Dr. Laurie Williams. If you have any questions or problems with the survey, please contact Neha Katira.

1. Think about all the partners you have ever pair programmed with. Consider which of these you have been the most compatible with. What qualities made you compatible with those partners? Check ALL that apply.
    a. same gender
    b. different gender
    c. he/she was of a similar skill level as you
    d. he/she had a higher skill level than you
    e. he/she had a lower skill level than you
    f. personality mismatch
    g. his/her sense of humor
    h. similar work ethic
    i. similar project management skills (e.g. you both wanted to start early, you both wanted to start at the last minute)
    j. similar work patterns (work at night, work on weekend, work during the day)
    k. he/she was punctual for your meetings
    l. similar age
    m. he/she was participator in all work
    n. just felt comfortably with that person
    o. same ethnic background
    p. same nationality
    q. other [please specify]

2. In what ways is it beneficial to have a compatible partner? Check ALL that apply.
    a. we are more productive
    b. we produce higher quality code
    c. we learn more from each other
    d. it is more enjoyable
    e. other [please specify]

3. Think about all the partners you have ever pair programmed with. Consider which of these you have been the least compatible with. What qualities made you incompatible with those partners? Check ALL that apply.
    a. same gender
    b. different gender
    c. he/she was of a similar skill level as you
    d. he/she had a higher skill level than you
    e. personality mismatch

f. he/she had no sense of humor
g. he/she was too talkative
h. he/she did not talk enough
i. differences in work ethic
j. difference in project management skills (e.g. you wanted to start early, he/she wanted to start last minute)
k. body odor
l. breath problems
m. dissimilar work patterns (work at night, he/she wants to work during the day)
n. he/she was not punctual for your meetings
o. age differences
p. he/she asked personal questions
q. he/she was not participatory in work and/or was easily bored
r. different ethnic background
s. different nationality
t. he/she made you feel inferior
u. language problems
v. other [please specify]

4. What setbacks do you face when you are incompatible with your partner? Check ALL that apply.
   a. we are less productive
   b. our code is not of as high quality
   c. we miss learning opportunities
   d. it is less enjoyable
   e. other [please specify]

5. How long have you been pair programming?
   a. less than 6 moths
   b. between 6 months and one year
   c. between one and two years
   d. more than two years

6. On average, what percentage of the day do you pair program?
   a. less than 10%
   b. between 10% and 25%
   c. between 25% and 50%
   d. between 50% and 75%
   e. more than 75%

7. Do you enjoy pair programming?
   a. Yes
   b. No

7a. If yes, why?
   a. I enjoy the social aspects
   b. We produce higher quality products
   c. We are more productive
   d. I learn a lot from my partners
   e. We avoid long debugging sessions
   f. Other [please specify]

7b. If no, why not?
   a. I like to work alone
   b. I often do not get along with my partner
   c. I feel like I am teaching my partners all the time
   d. I get more work done when I work alone
   e. I produce high quality code on my own
   f. I don't feel as much of a sense of accomplishment
   g. Other [please specify]

8. What is your country of origin?

9. In which country do you presently work?

Appendix D: Survey Comments

Qualities of the most compatible partners

self-motivated:

- "My most compatible partner also had a good drive to get things done, which really helps at 4 am."
- "He liked to lay clear core groundwork for the project before discussing the design."

mutual-respect:

- Having a "mutual respect" for the partner seems to be important when ranking the qualities of compatible partners.
- "humble" person

good communicator:

- "I felt most compatible with those who can communicate their approach, or better yet, multiple potential approaches. It really helps to have a partner able to *think out loud*".
- "Able to communicate well, was willing to listen even when they had their own idea of how to do something."
- "Communication - The best partners are good communicators."
- "Effective communicator, willing to be wrong and learn from me, willing to tell me I'm wrong and teach me."
- "Same native language resulting in no trouble communicating."
- "An ability to verbalize ideas about the task under development."
- "We felt comfortable enough to tell the other how we really felt about the project, code, design, process, and level of exhaustion."

ego:

- "Had a strong (or weak) ego. Medium egos are the worst."

complimentary skill levels:

- "complimentary skill levels (both of high skill, but in different areas of expertise)."
- "He/she had complementary skills and a desire to succeed."
- "Complementary skills – i.e. my partner is strong where I am weak and vice-versa."
- "Complementary skills are also important for pairing. That the pairs are knowledgeable about different aspects of the code or different techniques."
- "differing technical backgrounds".

similar skill level:

- "Skill level not to diverse (very senior and very new) also knowledge of product is important (right skills)."
- "There was enough overlap in skills that we could communicate and enough difference that we were better as a team."
- "The skills and experience were at a similar level, but our strengths were in quite complementary areas. We had some overlap of experience, but also lots of differences in our work history. He was strong in OO analysis/design and initial coding. I am strong in data analysis, database design and maintenance coding."
- "Good critical thinking skills - I find this much more important the general technical skills. With critical thinking skills, my pair can learn from or teach me. With poor critical thinking skills we're much more likely to be frustrated about the skill gaps between us."
- " Similar technical approach, for example balance between experimentation and up-front design."
- "He or she had a clue."

different project management skills:

- "Different project management skills - this way we have a good blend of things and it helps make better decisions."
- "Dissimilar work patterns/ project management skills have worked for me... "

similar work drive:

- "Similar drive (work ethic) - it is very critical for the compatible partner to have a similar work drive."
- "We were both goal oriented and like minded: which means that we agree on coding style, design approaches, best practices, and also that we can communicate well and can *read* each other's non-verbal clues."
- "Similar goals for the work session"
- "Similar programming style (e.g. OO vs. procedural, strong-typing vs. weak-typing, checked exceptions vs. unchecked exceptions, etc.)"
- "Same thought process and design thinking"
- "Similar ideas about what is good code, how much testing is wanted, how to go about the work. That we solve the problems together, or resolve them together."
- "Able to see the same vision for the work being performed"
- "Similar beliefs about *good code*. Knowledge of the project and/or domain and/or tools."
- "Both of us preferred staying on the cutting edge of technology."
- "Similar aggressive attitude in designing brute-force algorithms."
- "Similar approach to the problem/bug. It's difficult if you want to check something but they want to go off in a different direction."
- "Same speed at grasping difficult concepts".

adaptable:

- "Person was adaptable to explore items that came up during the session and had an interest in solving the problem."
- "Did not have an ego - e.g. was prepared to try new things or abandon things they had worked to try a different way or learn a new technique."
- "Both of us have a willingness and desire to produce better, more, faster and to make XP work. We still have conflict, but have each checked our ego's at the door."
- "Someone who isn't greedy with the keyboard".

open-minded:

- "I always liked partners who were open minded and believed in team spirit."
- "I've been compatible with everyone I've paired with, except for people with what I consider a *bad attitude.*"
- "Open mindedness, no big egos…"
- "They remained open minded and listened to all my suggestions."
- "Desire to experiment and learn is very important to me."
- "Willing to listen, patient, and willing to share the keyboard."
- "I have had many enjoyable and productive pairs over the years and have really only had major difficulty with one individual. He was against pair programming from the start and seemed inclined to prove his point by making it difficult for others to pair with him. It wasn't so much a personal-compatibility problem as it was a refusal-to-try-something-new problem."
- "Willingness to scrap their plan and open to new ideas."

sharing knowledge:

- "Attitude to share knowledge…"
- "Patient, didn't rush. Explained things. Didn't force the issue. Took things slow. Program was not only left in a working state (of course), but my understanding was too. Should mention that I have paired with Robert Martin & Dave Astels. The main person that I have paired with is definitely more knowledgeable, but tries too hard. Sometimes, I have to put my foot down, and say this is the best solution for now, i.e. no design pattern, maybe not the best OO principals applied, but I understand it."
- "Willingness to share his *knowledge nugget*."

existing relationship:

- "Existing relationship (friendship) long established, making comfort in pair programming very natural."
- "None of these are important at all, other than maybe *just felt comfortable*.

uncategorized:

- "He/she had a passion for design and programming."

- "He liked to overhaul the coding options extensively before coming to a decision."
- "Believed that there is no *I in TEAM*."
- "I like someone who is:
  * A quick thinker
  * A fast typer
  * Willing to modify their ways and learn from me (hey, I'm good!)"

Q**ualities of least compatible partners**.

complaining:

- "He overcomplicated each task and did more complaining than working towards a solution."
- "He/she had a bad temper."
- "Someone who usually resolved problems by asking the coach - which I felt showed disrespect of me. I later learned that this person had the illness called depression. I feel like generalizing this to saying when someone is unhappy, pair programming may be difficult."
- "*Arrogance*: Arrogant people make for poor partners."
- "One time I have not wanted to pair with someone was when I had to work with this ill-mannered person, who was loud, would swear when angry, and angry often. He was a gaijin, foreigner, working in Japan, and apparently not too happy about it."
- "*Mannerisms*: one guy would fidget a lot which was distracting to me; I could eventually get used to it, but it was difficult."

not have the same mental image of the problem:

- "Not engaged in the problem. If the person does not have the same mental image of the problem at hand, then it is not pair programming. It may be training, or mentoring, and this may even be useful, but it is not pair programming."
- "Different ideas about programming style, different senses of *what smells*."
- "Difficult if person doesn't *get* object oriented abstraction, and instead favors copy and paste reuse. Difficult if person does not see the bigger picture."
- "Dissimilar programming style (e.g. OO vs. procedural, strong-typing vs. weak-typing, checked exceptions vs. unchecked exceptions, and the like)"
- "Good at programming but bad programming habits. Not realize importance of naming conventions. Doesn't grasp what *simplicity* is. Too good at old style programming or too new."
- "We had different problems solving approaches. As the Driver worked through a problem, it was hard for the Observer to follow along and give input/feedback on where there might be holes in the code or in tests. Frustration came in when we could never get to an agreement on the problem and how to solve it."
- "Or, in other situations, there were too many differences of opinion regarding

technical approaches (e.g. relational database versus other persistence layers)."

lack of right background:

- "Rather than skill level, it might be more accurate to say experience level/familiarity with our code base. I'm working on a legacy code base, which is in pretty bad shape. Because it is poorly factored, it's hard to understand, and even though my partner is a good programmer, his lack of background made things very slowly."
- "Different pair programming experience."
- "It's not the skill level as much as the speed to learn and program…"
- "Different level of rigor (different inclination toward error checking and exception handling)."
- "Asked too many insignificant work-related questions. Questioned too many coding decisions."
- "Did not understand the problems we were attempting to solve programmatically and had a weak grasp of math skills needed to create innovative results."

not flexible/attitude problems:

- "No willingness to compromise"
- "He/she preferred to work alone", "He did not do a good job of listening to alternative approaches - it was his way or the highway."
- "Disinterest in pairing or any of its benefits; being in it for the rush and exhilaration of slamming out code, not for the customer, though still theorizing about the *user* in one's imagination; having a sort of attention deficit problem where any slowdown causes distraction and frustration - in other words, not wanting to interrupt the stream of consciousness for anything, or feeling that their code should be exempt from scrutiny at least until a few more thoughts get completed... I believe that there are some people who would need nothing short of a personality change operation to be effective in a pair programming situation!"
- "I did not like to pair with someone who had no interest in the work. I was particularly frustrated when my partner, who was not serious at work."
- "Partner pushed own ideas without consideration for mine."
- "Took control of the computer too often. Always thought he knew what to do, but really didn't. Disregarded my ideas. Traversed unnecessary exploited paths. Wasted time."
- "He/she took sole access of the keyboard."
- "Resistant to change."
- "He had ego issues, attitude problems, and liked taking rewards."
- "Specifically, this person had problems working with other people. Too much, *We do it my way or the highway*, which lead to many, many arguments and not much getting done, just loads and loads of stress."
- "Someone who felt that my development practices were foolish."

- "Has own view/ideas/plans, but unwilling to share or listen, does not do test first, does not see the big picture."
- "He wanted to do all the work himself."
- "I feel, it is the attitude that matters. There are times when a partner with lower skills tend to tag along without making any effort to learn. If we assume the ideal situation then all programmers tend to move towards improving themselves then pair programming works, but I find that this is not the case a lot of times. I would stress on the attitude of the pairs. After not all of us want to become pragmatic programmers."
- "Lack of openness to experimentation, trying something new, try-and-fail, tendency toward analytic approach (too large steps)."
- "Some people's personality/ego make them bad pairs. They think they always know better than their partner even if they are of a lower skill level. Also, and this is important, they do not have the social skills to make compromises wherein both members feel comfortable with the final decision."
- "He/she made me feel inferior: incompetent"
- "General social issues, mainly the inability to express oneself in a positive manner, i.e. that's a stupid way to implement ...; what the hell is ...?"
- "The main problem is respect. If the person I am working with constantly looks down on other people's (or my own) code then it is a serious problem."
- "Wanted to stick to painting by the numbers. Was not open to new ideas."

inability to communicate:

- "He could not communicate well -- we couldn't *read* each other's non-verbal clues."
- "He/she had poor communication skills."
- "Those simply not willing to collaborate, meaning work together on programming, especially if they are not able to or willing to communicate, are least compatible."

difficult to motivate:

- "He/she was intimidated by me - needed to be coaxed to express opinions or push back on things.
- "It was difficult to motivate the other person, he felt very threatened by my greater knowledge"
- "I had to work hard to avoid fostering feelings of inferiority in pair partner."
- "Short attention span made pairing difficult unless they were the one with the keyboard."

breath problems/body odor:

- "I have not personally experienced a partner with body or breath issues, but I think it would be difficult to work with."
- "My partner was a smoker and had body odor."

uncategorized:

- "The person did not know how to type very well."
- "Physically uncomfortable working environment, i.e. differences in keyboard & desk preference!"
- "Anything that prevents you from sharing goals you can move forward."
- "He thought that he could fill in the blanks for me."
- "Ambiguous working relationship."
- "Extremely daunting in attitude."
- "Non-communicative work culture."
- "Assigned himself as a de facto team lead."
- "This feels like therapy. Forgive my indulgence:
  Case 1: He had a completely different (non-xp) way of coding. It was defensive, huge blocks of coding, up-front design, slow. It drove me mad. I felt bad afterwards about how I handled it.
  Case 2: He was a 40 something *architect*, skeptical of XP, and wanted to design everything up-front. He wouldn't just code a bit and get some feedback. I lost my temper. I shouldn't have.
  Case 3: He suffered from clinical depression and had bad concentration problems. He was a really slow typer also. He took out his frustrations on the code I had written and said it was a tangled mess (the ultimate insult). I felt stupid."
- "Distractions due to other work assignments"

Benefits of Working with a Compatible Partner

time management:

- "Our project was completed on the tentative timeline (sometimes early)."
- "It keeps you on track. Not as many side trips. It also allows questions to be asked quicker."
- "I don't know if I am more productive but I get a kick in the ass so I don't fall asleep, or do other things like surfing on the web."
- "Code gets completed on schedule and budget."

two heads are better than one:

- "We can tackle a bigger problem. Programming is like a juggler. A programmer keeps ideas and data structures in their head as they make decisions on how to structure the code. It is only possible for a fixed amount of stuff to be in the air at one time. Properly pairing, two heads are holding up the same problem, which means that when one person drops a concept, the other can pick it up before it falls. I have tackled problems quite quickly pairing that I made little or no headway on my own, and neither did the person that I was pairing with."
- "I think it is probably a combination of all of the options in the question. When it

works, you're *in the flow*"
- "Possible to do things not otherwise possible."
- "Fewer design decisions come from individuals without any discussion."
- "Less time spent in pointless discussion/teaching. Able to dive in to productive work. I feel best about programming when I'm getting useful work done."
- "*Courage*: being two makes it easier to take the right but tough choices."
- "Things never look tough."
- "Immediately had up to two perspectives on any given problem as it was introduced."

communication:
- "Compatible partners communicate well with each other (verbal and non-verbal)"

comfortable:
- "We are less concerned about making mistakes in front of one another"
- "Pair programming is good as long as you have a good partner. If that does not happen, you can uncheck all the options above."
- "We have a *synergy*"
- "You just don't notice how time flies."

less stressful:
- "*Less stress*: you know better what must be done and who should do that."
- "Less frustration"
- "One can take vacation and the other is there to answer questions."
- "It's less stressful when you like or are comfortable with your partner. When you're pairing, you're always *on* so it can be very stressful."

stay focused:
- "Although I might not be learning anything from my partner, having a sharp partner keeps me sharp somehow."
- "Related to the productivity: We are more focused"
- "Compatible partners spend less time arguing about the approach to take and can focus more on solving the problem. Incompatible pairs don't trust each other as well, so they have to question all of their partner's decisions."
- "We are less easily distracted by interruptions (e.g. other people asking questions, email.) We are more easily distracted by off-topic conversations about which we are both interested."

knowledge sharing:
- "Share not only technical skill, but also domain knowledge, which I think is very important too."
- "Knowledge is shared among the team."

incompatible partner:

- "I prefer incompatible partners."
- "Having a non-compatible partner, it becomes a daily battle to do thing right."

Setbacks of Working with an Incompatible Partner

differences in skill:

- "The worst experiences for me are *always* with less skilled programmers. I feel that I have to stop every two minutes to explain the least little thing, and this seriously impedes any sort of forward progress."
- "If skill gap is too great, or one partner's skill is not at a sufficient level, the other person spends more time teaching and explaining, instead of concentrating on the task at hand."

pairing time:

- "I become more tense and less willing to continue. I want to shorten the sessions and often concede just to get to the end of the session"
- "We just could not work and achieve our goals. We got caught in silly issues and wasted time."
- "I find that pairing with the same person for more than six months (because it's a 2-person team) causes what I call *pair fatigue*. There's less to learn from each other. You get tired of each other. It's less fun. You really need to split up or mix it up. You can do XP with less than four people, but less comfortably. That's a tradeoff you sometimes have to consider, because staffing is expensive."
- "Generally it makes for a long painful day, and I'm less inclined to look forward to tomorrow's pairings."
- "I get very tired. It's very intense."
- "Missed deadlines, client was disappointed, and lowered my work enjoyment"
- "Concerning code quality, we tend to just get tasks done and over with so that the pairing can end sooner. If that isn't possible the pairing tends to switch to another partner midstream."

burden of work:

- "One person does all of the work, just as if they were working alone, and the other watches, is disengaged, and is bored. Quality does not suffer as long as the person doing the majority of the coding maintains their personal standards."
- "I have to do all the work while my partner sits in a daze. Or I have to tell my partner what to do and correct her work."
- "And both me and my partner will feel that extra work is required to fix the other person's code..."

affects work produced:

- "Re-factoring can be more of a re-write if the code is re-factored by someone with a different approach to the problem."

- "Sometimes working too closely with someone may be worse overall (less productive, etc) than working individually if the people don't get on. I think of it as two oppositely charged magnets, there's no resistance until they get close to each other."
- "I land up with the worse design/code."

no pair programming:

- "We stop pairing."
- "The pair programming doesn't occur."
- "Partners polarize tasks. i.e. one develops, one cleans up (maintains standards). Less refactoring."
- "We don't really end up *pair programming*. The tendency is to split up the work and do it separately."
- "[Pairing] does not last. We stop pairing and the knowledge of that person's code stove-pipes."

impacts team performance:

- "Bad pairs argue more; I very much dislike disagreements on our team so this leads to extra stress/tension for all team members. Also, bad pairs are generally bad with all other team members so you get into the situation where the rest of the team argues about who has to pair with them next. It also leads to the team not switching pairs often enough."
- "It also has negative impact on team performance and team culture."

work ethic differences:

- "Less reliable, i.e. partner did not share sense of urgency/dedication in mission."
- "We were not able to hit goals at the same time and he could not recognize a project that was nearing completeness, so he attempted to keep adding useless code, extending the project needlessly."

stressful:

- "It becomes frustrating to work with that particular person."
- "It's stressful."
- "There is no sense of accomplishment and you're tremendously tired."

uncategorized:

- "It's more difficult to think through difficult algorithms when working w/ someone else."
- "None. It's still all good for the code."

Do you enjoy pair programming? If yes, why?

stay focused:

- "Spur others on and keep focused"
- "We work longer without breaks. It is only possible to keep the mental train of thought for a specific period before a break is necessary. Pairing, this is significantly longer, which makes you mentally tired at the end of the day, but satisfied by the amount that got done. (which is more that what I would get done alone in two days by the way)."
- "It helps keep people interested in solving problems."
- "In all it is a lovely experience. Our productivity doubles when we work in pairs and the attention to detail in every aspect is very high."
- "Things seem to have more direction. I can brainstorm and be creative while my partner can keep me grounded, and vice-versa."
- "Allows me to focus better on the problem. When I am pairing, I tend not to take unnecessary breaks."
- "Good to get focus and flow state of mind more quickly."
- "We keep focus at the right level."

knowledge sharing:

- "Knowledge of the entire application is shared across the whole team."
- "Ideas are free flowing. There is no one time where we can say that we are short of ideas. Especially while designing applications which demand precise and pragmatic approach; one of us wears the thinking cap to design and the other would take the role of a major critic. This way we are able to sort of most of the issues that would otherwise go unnoticed."
- "Knowledge transfer is better."
- "I usually work in a lead role. Pairing with a competent partner allows me to get a new project up and running while at the same time training my replacement. Once the partner has learned everything in my head about the project, I can safely leave things in his/her hands and move on to something else that requires my attention."
- "Learn more by GIVING as well as FROM partner."
- "I like to teach too."
- "Natural knowledge spread occurs with pair programming."
- "Communication is very important between team's members."
- "Able to train others (learning goes both ways)."
- "Code knowledge is shared. We avoid the hit by a bus syndrome."
- "The communication and dialog during pairing I find to be invaluable. There are no pockets of information and if anyone on the team learns a new trick or about a new library the rest of the team finds out about it quickly."
- "Knowledge transfer occurs naturally."

work time:

- "Time flies"
- "Long code review and design meetings are eliminated. Everyone on the team

knows everyone else on the team is *doing their job*, and that considerably reduces the finger-pointing and political maneuvering."

- "Design issues can be discussed immediately with a partner when they arise during coding. This avoids having to postpone work until a later time after it's been possible to discuss the design with someone."

sense of ownership:

- "It builds a powerful sense of community."
- "Collective code ownership: knowing much more about the code, being able to work on whatever part I need to, and yet knowing I'm not the bottleneck for any of it."
- "Shared responsibility for the code; the knowledge is spread out so others can work on it as well rather than just *the owner*."
- "Supports collective ownership, team-building, code-review."

two heads are better than one:

- "Highly collaborative; I enjoy getting to know how my pairing partner thinks."
- "There are two things I've noticed in particular about pair programming that make me want to do it all the time. First, when I'm stuck and don't know how to get around a problem, working with someone (or switching partners) is a good way to get back on track. Second, it's much easier to stay focused on getting the work done when I'm working with someone else."
- "Two heads are better than one."
- "Used to solve complex problems or find bugs that a single person might miss."
- "Being able to discuss design and trade off issues with someone [in] real time."
- "Sometimes my partner and I have different ideas of what the possible solution to a problem might be. It is not unusual that we find a third solution, which is superior to the other ideas. I call this joint learning. It is not that we learn from each other, but that we inspire each other to be more creative and to find/discover completely new solutions in coding, design, testing, architecting."

effects on work produced:

- "We end up with consistent code, consistency of style, naming, idiom, and so forth."
- "Combined with test first, it makes all problems smaller and simpler, so each day is more fun."
- "It keeps me moving in a forward direction. It reduces paralysis when paralysis is the result of not knowing what direction to choose (on decisions both large and small)."
- "Get better module tests and instant code reviews."
- "Faster - less need for formal inspections."
- "Easy finding partner's errors."
- "Currently our two primary competitors have 5-10 times the resources we do. To

compete effectively (or even at all) with that type of imbalance, we need to have our team running as efficiently as possible. By pairing, we can create better code faster with fewer resources than our competitors. The knowledge transfer between our team members is amazing, and has allowed us to each become top-notch programmers in a very short time. Spending eight hours a day paired with smart people has made me much better at what I do."

effects on individuals pairing:

- "Pairing provides me with additional confidence in the quality of my development. In absence of pairing, I find myself second-guessing the work that I produce."
- "I just plain get mired and depressed without a partner, and I start pulling my hair out. I guess it's my personality."
- "I kind of learn how to get along well with different people and improve my skills in that aspect."
- "It helps me to see what my abilities and inabilities are by comparison with other people. Without pair programming, I just guess and worry that I am inadequate."
- "I have ADD. This is the only way I feel I can be productive, unfortunately I am not in a pair programming situation right now."
- "I feel more motivated."
- "It seems like a faster way to reach my goals (an effective programmer) and other's goals at the same time. Please note that from my experience some tasks are suitable for pair programming, some are not."
- "As I don't like programming in itself, but I'm very strong in finding problems/ bugs in an existing code, the collaboration with a more junior colleague allow us to produce high level results quite fast. The only drawback is that the learning is mostly one-way (although I learn from mistakes I see too)."

uncategorized:

- "When done well, pair programming is enjoyable and productive. When poorly matched, it can be less enjoyable and productive than working alone."
- "Our conversations are often overheard in the open workspace, and others voluntarily join in to clear misconceptions or contribute. If we were to work solo, the conversations wouldn't happen."
- "Contributes to team cohesiveness."

Do you enjoy pair programming? If no, why not?

personality of the programmers:

- "I am an extreme introvert, and I find that working with a partner for more than an hour or so at a time is totally exhausting. I often need to be by myself for a while to recharge. I thought I'd mention this as an *if not* comment even though I DO enjoy pair programming."

- "Sometimes, I'm afraid of looking a fool."
- "Exposes my insecurities."
- "I don't feel like I am learning Java; there's always someone to fall back on. Strangely, I can either think or talk but I can't do the two things together. It's very tiring."
- "Sometimes it is nice to just get away from the social aspect and get some work done. It may not always be the best solution, but it's a good breather."

work environment:

- "Unsuccessful pair-programming can come from the desktop arrangement. Working with someone else on his computer (not in a neutral environment) can be difficult. You do not feel at home there. You tend to watch more and participate less. Difference in screen resolution and so one can also be annoying. Spare room around the computer is important too. If there is just room for one person, it gets difficult to enjoy it."

like working alone:

- "At times I like to work alone...."
- "I don't enjoy programming period."
- "I feel that good code reviews and unit tests can accomplish better quality code. Pair programming is very intense and can be mind numbing when done for long periods of time. When your mind becomes that tired its easier to overlook different things. I also believe that two people can convince themselves that something is right with much less scrutiny. When working alone you tend to think through problems in more depth."

uncategorized:

- "No partner available."
- "A bad pairing relationship can degenerate into lengthy and fruitless discussion even on small points, or other problems, which make the situation worse than no pair."
- "Partner was reluctant to share, and believed in the proverbial *Knowledge is Power* mentality thinking that they will lose some real or imagined advantage.
- "If the pair is working properly then it is fine, else all the negative options in the question apply."

Appendix E – Open ended responses

"Okay, I done it. However, the survey presents the bias that compatibility is good for pair programming. Socially, I'm married to someone that no dating service would have ever aligned me with. Technically, and within a secure team environment, I prefer incompatible pairs. Every incompatibility, from poor English skills to the desire to fight my every design pattern, all redound only to the greater glory of the code. And to a 6.5 hour work day."

"I think that there is no silver bullet for compatibility; however, a very important consideration is that the pairs have a *respect* for each other and each other's ability. Understand where A is stronger than B and vice-versa. Learn from the each other as you go to get up to the other's skill level at whatever. Leave your ego at the door, it will surely kill any collaboration, which are attempted. Note, respect has to be earned. Leave your reputation at the door, too (but that's part of ego, isn't it?)."

"The skills should complement each other. The ivory tower *grand planner* types should be paired with the anal-retentive *let's worry about every detail before we write a line of code* types. (Exaggeration for effect). If they don't kill each other during the first week of collaboration, they usually wind up doing great work together."

"Then the management says, 'Okay, we'll give this *pair programming* idea a one-week trial, to see how it goes...'"

"The truth is there are far more complicated processes going on (because our brain is far more complicated), and when it comes to the cognition and interpersonal interactions of a software developer or engineer, where success during analysis, creativity and ingenuity are key aspects of the job, conceding to the simplicity of a catchy title can do more harm than good. Moreover, as a pair, the cognitive cohesion of the two minds and conceptual integrity of the tasks performed become crucial in predicting the productivity of the pair and the quality of their work. Pairs need to be identifying a synthesis of their views. A real life example might help me explain. Pair programming is used at one of my clients (they call it pair engineering). I found that in the beginning, social cognition and social perception, along with some *Groupthink* phenomenon could influence the outcomes. One pair consisted of a couple of experienced developers with pair programming that is a good example of this.

Without going into the details of the two developers, what happened was a tendency toward *consensus*. Anyone who studies social psychology and group processes would know that such a tendency is the opposite of *critical thinking* - the necessary paradigm to make pair programming effective. The motivations behind some of the rules in the abstract of the cited article play into the motivation toward seeking consensus.

What made it worse was they believed that the pair's decisions were right. The result was wasted time during pairing, and the two only discovered serious design and implementation problems when they were not pairing - later after the mistakes were made. In order for pair programming to be worth the effort, we must go beyond computer science. It

isn't difficult and it isn't unknown - we simply need to learn from other disciplines and apply the principles and techniques they've known for decades."

"There are various things that can be done to reduce the negative impact of pair programming: Make sure pairs rotate a lot. Have open discussion on pairing- what works - what doesn't. Coach may have individual coaching on communications theory and practice."

"I'm supposedly a domineering personality. At least, I am loud, very smart. And I have had to learn how to give others a chance to talk, e.g. in meetings. I know that I have enjoyed my pair programming experiences. Ideally I would let my pairing partners speak for themselves, but I don't think any of them are on this mailing list – and, even if they were, they are usually not the sort of person who posts. So, what I will do is record my impressions of their feelings; I will also send this email to some of my partners. Perhaps they will respond, although possibly point to point. I've paired most extensively with

- T - a very shy and quiet guy
- R - not so quiet
- H - short pairing, somewhat quiet
- K - short pairing, not so quiet
- M - probably more domineering than me.

In all of these situations it was just two people pairing. Not part of a bigger XP team.

Case T: T was very initially very reluctant to pair with me. He was reluctant about pairing over all, and he expressed reluctance because he was worried that I would dominate. I got T to agree to pair with me on condition that, after two weeks, we would stop and go back to non-paired programming if he did not like it. After two weeks, he said that he liked it. Moreover, after two months, *I* wanted to take a break from pair programming - probably just a mood swing, "Is this really working?" T then persuaded me to stay doing this. I gather from this that T actually liked pair programming. T did, occasionally, call me when I was domineering: "Hey, Andy, it's my turn!" We probably talked loudly at each other more than twice over a six month period, but less than six times. I think the key was that we prearranged signals – "My turn", "Hold on" – and that we took efforts to be conscious about the pair dynamic. It probably also did not hurt that I, the *dominant* personality, was in the position of trying to sell XP and Pair Programming. Also, although I am loud, I am not a type A personality on the classic psychobabble charts. I rate as an *aggressive bridge builder*. Taking time, regularly, to ask "how is it going?"; "how are we doing" was important.

Case R: Pairing with R was similar. R was less shy than T, so R had no problem stopping me from domineering. R and I evolved an interesting dynamic: I was the *expert* on the programming domain. Usually I would explain via mini-tutorials. Occasionally, I would ask R to just allow me to code for a few minutes, with him on looking. Often he understood things after seeing me sketch the code (or, as

often, the test), better than if I had provided the tutorial. Occasionally, R turned the tables on me, saying "I can see that you think you know what we should be doing so why don't you drive for some more time?". This was R's first experience with test-first. He had a habit of trying to write the code before the test. It was my role to drag us back to TDD.

Case H and K: Pairing with H and K was similar.

Case M: Pairing with M was most interesting. It was probably the least enjoyable pairing experience I have had -- but we got some really good work done, both of us feeling that the emergent design was better than what we would have done individually. Both M and I are alpha geeks, M probably more so than me. M and I have worked together over ten years, but this was our first pairing experience. We are also very good friends outside of work. Still are. M has a severe hearing problem. He relies on lip reading. A big source of frustration to me was realizing that M had not heard what I was saying, and was just assuming it was something very stupid and may sound totally irrelevant to the disucssion. Biggest tension: M is adamantly opposed to XP, adamantly opposed to shared monitor pairing, adamantly opposed to testing. "I test my code; I just don't automate the tests." M felt that my writing tests was just slowing him down. I resented being put in a role where I was writing the tests, and him the code. We fought over this constantly and were not comfortable at times too. But even with all of this, we came out with something pretty good. Eventually, I called off the pairing after a month or so, expecting to return to it after a break of a not pairing for a few weeks. I was hoping that would releive me. Although pairing with M was stressful, for both him and me, I would do it again – but I would be much happier if we were not pairing day in and day out for such a long time. E.g. if we could switch off partners. M doesn't want to pair with me again, but has paired with other, less domineering, people. My conclusion: pairing two dominant people can be stressful, but can also lead to good work. Interleave with other pairing."

"The short of it is personality does not dictate specific behavior, although it does both shift and mediate it. So those who are domineering can learn how to throttle their enthusiasm, provided their motivations/intent is compatible with the partner. If the partner is particularly concessive or timid, they can learn how to be more assertive. It is uncomfortable at first, but it is worth the personal growth. The short of it is what I have witnessed is the more cohesive the pair is, the more alike their interactive behavior becomes when they are pairing. The important thing to consider is that information that is not shared but should be, can be crucial to making the right shared decision every time."

"What I failed to mention is that if someone is having a difficult time with working closely with people at work, they likely have difficulty with relationships outside work as well. Thus personal growth as a pair programming partner can also improve relationships elsewhere because working closely with people for any reasons involves some of the same

phenomenon and principles."

" ...everybody had their own keyboard... and whoever can type the fastest wins? On the first project I had to start pairing full time I was not comfortable leaning over the same keyboard. Initially we set up two computers right next to each other running VNC between them. Bigger monitors and bigger fonts also help.
I am betting the survey will show that similar skill level is the most important factor in successful (enjoyable, productive) pairing. Obviously for me it is. I have not found domineering personalities to be a problem. Although if the person is also some kind of a lead and uses that position to impose his/her will, then you might as well not be pairing. It's much harder pairing with someone that's very quiet than someone that's domineering. Although even with quiet people, I've seen a colleague use great techniques to engage them.
e.g. ask questions like "What do you want to call this method", "Where should this class go", "What should this test do", etc."

"These arrangements seem practical but, in my opinion, in general are not necessary. I think the driver/navigator paradigm is what makes pair programming tick. Situations like right/vs./left-handedness, natural/vs./ergonomic keyboard/mouse preferences might warrant such elaborate mechanisms, but, in general, do we really need them? Most of the power of pair programming comes when a pair works as a unit with the brains/minds scaling different cognitive levels more fluidly since the other mind is there to depend on for not getting stuck. I am sure all of us have had the jitters when we are not in the driver's seat. But imagine the havoc that would be created if there were two sets of brakes and accelerators! Wouldn't the same apply to pairing?"

"The essence of pair programming is that you have two brains working together. As I'd like to see how two people could work in two difference locations and still pair (this is what I am getting out of your explanation). Although I am rarely opposed to trying new things, I would imagine this would be a challenging setup; one which, in my opinion, wouldn't warrant the effort if I had a team, which could work at the same location. I would approach it as an experiment and setup a handful of developers (four) to work in this way. Then evaluate it after the first couple of iterations, then compare the evaluation against the list of potential issues brainstormed while setting up. I wouldn't dive head first."

"I enjoy pair programming. I also like to work alone. Unfortunately, it is impossible to avoid the conclusion that people who like to work alone don't enjoy pair programming because the ONLY people who are able to say that they like to work alone will be the people who don't enjoy pair programming."

"Is a sense of humor ALWAYS a good thing? I could see having an incompatible partner who was incompatible because they were joking all the time and a two people team who were compatible because they never joked while coding (granted that wouldn't be me, but it seems likely)."

"*Compatibility*: You talk about higher and lower skill levels - but you do not discuss complimentary skill sets. The most fun I have is when I work with people who are of a similar skill level, but with a complimentary skill set. They can show me things in one area, and then I can show them some things in a different area than theirs'.

There are times to not pair program (i.e.: Tasks that are lethal to pair programming compatibility). In my experience there are tasks that you need to just go ahead and do. Pair programming adds no value, and in fact it can kill the partnership if you have two people working on this type of task. If people work a lot with this type of task, it will impact on the answers to the questions you present."