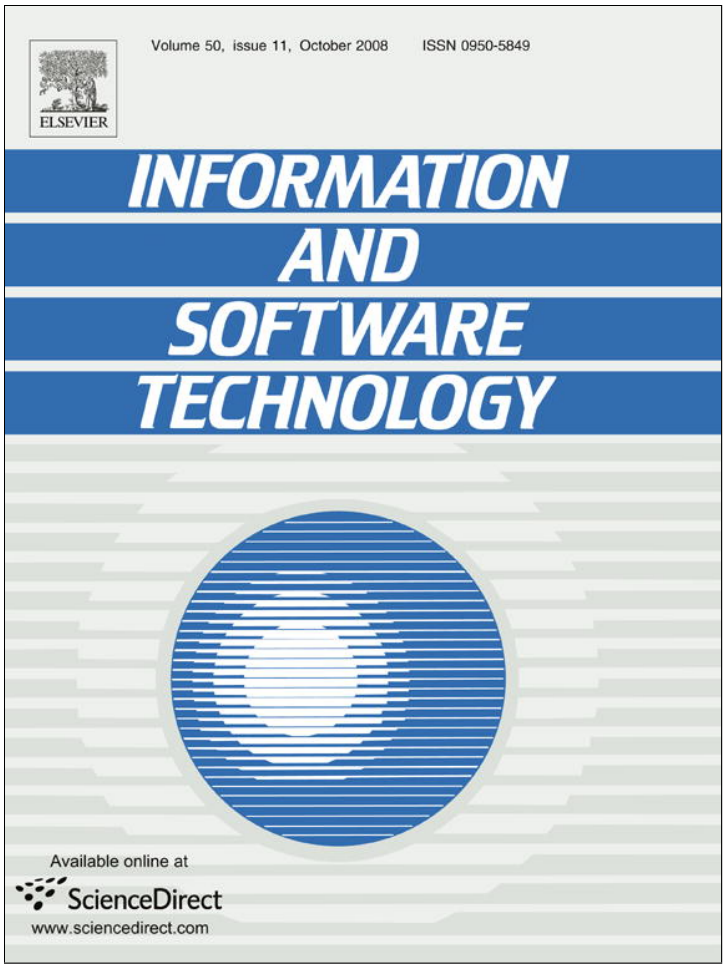
Volume 50, issue 11, October 2008
ISSN 0950-5849
ELSEVIER
INFORMATION AND SOFTWARE TECHNOLOGY
Available online at
ScienceDirect
www.sciencedirect.com

Available online at www.sciencedirect.com
ScienceDirect
Information and Software Technology 50 (2008) 1114–1126
INFORMATION AND SOFTWARE TECHNOLOGY
www.elsevier.com/locate/infsof

Exploring the underlying aspects of pair programming: The impact of personality

Kyungsub S. Choi [a,*], Fadi P. Deek [b], Il Im [c]

[a] *Faculty of Computer Information Systems, School of Business, Manhattan College, Riverdale, NY 10471-4098, USA*
[b] *Faculty of Information Systems, College of Computing Sciences, New Jersey Institute of Technology, 323 M.L. King Boulevard, University Heights, Newark, NJ 07102-1982, USA*
[c] *Faculty of Information Systems Department, School of Business, Yonsei University, 134 Shinchon-Dong, Seodaemoon-Ku, Seoul 120-749, Republic of Korea*

Received 2 July 2007; received in revised form 30 October 2007; accepted 2 November 2007
Available online 19 November 2007

Abstract

With the recent advent of agile software process methods, a number of seldom used and unorthodox practices have come to the forefront in the field of computer programming. One such practice is that of pair programming, which is characterized by two programmers sharing the same computer for collaborative programming purposes. The very nature of pair programming implies a psychological and social interaction between the participating programmers and thus brings into play a unique element that we do not see with the conventional individual programming model. This paper focuses on the effects that one of these psychosocial factors, a programmer's personality type, may have on the pair programming environment. In this study, a group of university students, 68 undergraduate students and 60 master's degree graduate students, each of whom had been personality type profiled using the Myers–Briggs Type Indicator (MBTI) model, was split into three sub-groups. One group consisted of subjects who were alike in MBTI type. Another group consisted of subjects who were opposite to each other in MBTI type, and the last group was comprised of subjects who were diverse – partially alike and partially opposite – in MBTI type. Through two pair programming sessions, the pairs in each group were assessed for their output, in code productivity. The result showed that the sub-group of subjects who were diverse in MBTI type exhibited higher productivity than both alike and opposite groups. In a comparison between alike and opposite groups, the productivity of the opposite group was greater than that of the alike group.
© 2007 Elsevier B.V. All rights reserved.

Keywords: Pair programming; Team programming; Collaborative programming; Myers–Briggs Type Indicator; Personality

1. Introduction

Before discussing pair programming, a brief introduction of agile software process paradigms is relevant due to their close association with pair programming. Agile software process paradigms, or the "agile" movement [3,12,13], can be described as a collection of unorthodox software development paradigms that are characterized as adaptive, result-oriented, and human-centered. The focus is on producing premium quality software in the shortest time possible. The blazing speed of electronic commerce, ubiquitous computing, the fierce competition among the electronic and software manufacturers, and the growing user base of information systems literate consumers are only a few of the pressures driving today's software engineering community [4,5]. These pressures are forcing software professionals to look for answers outside of traditional software process paradigms. Except for large legacy systems or mission critical projects that are for more established paradigms such as the waterfall model [34], one option for smaller software projects is the new agile approach. The attributes of agile software process paradigms, including short development time, moving user requirements, frequent releases, and customer focus, read-

[*] Corresponding author. Tel.: +1 718 862 7309; fax: +1 718 862 8032.
E-mail addresses: kyungsub.choi@manhattan.edu (K.S. Choi), fadi.deek@njit.edu (F.P. Deek), il.im@yonsei.ac.kr (I. Im).

0950-5849/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.infsof.2007.11.002

ily embrace today's volatile business and software ecosystem.

Among many agile software process paradigms, the most popular one is extreme programming, or XP [7]. XP is well-suited for "risky projects with dynamic requirements" and mainly geared for small to mid-size application development. Software process stages that are considered unnecessary are eliminated, freeing programmers from documentation and other overhead chores. XP operates on 12 core principles: the planning game, continuous testing, on-site customer, small releases, refactoring, a 40-h work week, system metaphor, continuous integration, simple design, collective code ownership, coding standards, and pair programming. Beck [7] states that the 12 core practices yield the full benefit of XP only when they are used together. As dramatic as these practices are, only a few are used together in real world applications [21]. But even such a partial adoption may be a good introduction to those who are skeptical of XP.

By most software engineering standards, the techniques used in XP are viewed as unconventional. For example, in traditional paradigms, the software development process focuses on formal software requirements documentation. But in XP practice, the software development process focuses on feedback from an on-site customer, a customer representative who remains on-site throughout the development project. Despite these unorthodox principles, XP yields promising results [28,40,42]. XP's no documentation, no formal requirement engineering, no up-front design, and other anecdotal features are under debate [11,20], but often lead to high productivity in the form of frequent releases, high quality and fewer code errors [14].

The focus of this study, pair programming, is defined as a programming activity where two individuals sit next to each other sharing one keyboard and one display. This can also be done virtually with two programmers in geographically different locations sharing one common view of code development and taking turns programming [37]. While one person, who is called the "driver," is coding, the other person, who is called the "navigator," does real-time code review [41]. Each programmer takes a turn being the "driver" and the "navigator." The active collaboration in designing, coding, and reviewing thus eliminates the overhead cost of a more formal code review.

2. Background

Many previous empirical studies focused on proving the value and efficiency of pair programming. From its first real application [1], pair programming has been compared to individual programming [28] and tested for code quality [40], programming experience [25], and economic value [14]. Many of these explicit attributes continue to be the domain for many studies. However, one should assume that the inherent nature of pair programming involves psychological and social interaction between the programmers [15]. Although the values of the psychology and cognitive sciences have been appreciated and respected [2,39], it has never been in the spotlight as one of the main programming research domains. Riding on the coattails of the human-centered approach of the agile movement, however, the psychosocial aspect of programming is starting to receive more attention. Further studies on the psychosocial aspect of pair programming may have an even greater impact, due to the high degree of human interaction between the programmers.

To study the impact that personality type has on a pair programming environment, we need to find an appropriate personality profile assessment model to characterize personality types. Many business organizations currently profile new employees by using a popular personality assessment model called the Myers–Briggs Type Indicator (MBTI). Although MBTI is one of the commonly chosen personality assessment variables among organizational and team building studies [8,9,23,35], analysis of the extensive impact of MBTI needs to be continued [29].

MBTI measures a person's preferences using four basic scales with opposite poles. The four scales are: extraversion/introversion, sensate/intuitive, thinking/feeling, and judging/perceiving. People who prefer extraversion tend to focus on the external world of people and things, whereas people who prefer introversion tend to focus on the inner world of ideas and impressions. People who prefer sensing tend to focus on the present and on concrete information gained from their senses, and people who prefer intuition tend to focus on the future, with a view toward patterns and possibilities. People who prefer thinking tend to base their decisions primarily on logic and on objective analysis of cause and effect. People who prefer feeling tend to base their decisions primarily on values and on subjective evaluation of people-centered concerns. Lastly, people who prefer judging tend to like a planned and organized approach to life and prefer to have things settled, whereas people who prefer perceiving tend to like a flexible and spontaneous approach to life.

The various combinations of these preferences result in a total of 16 personality types and are typically denoted by four letters to represent a person's tendencies on the four scales. For example, ENTJ stands for Extroversion, Intuition, Thinking, and Judging. This does not mean that a person possesses only four preferences, but that the four preferences show a greater presence than their counterparts. For example, Fig. 1 is a typical profile report from Consulting Psychologists Press, the MBTI authority organization. In this example, E is showing a greater presence, on a moderate level, over its counterpart, I. N is showing a greater presence, on a very clear level, over its counterpart, S. T is showing a greater presence, on a very clear level, over its counterpart, F. Lastly, J is showing a greater presence, on a moderate level, over its counterpart, P.

From the four active preferences, the person is most comfortable with one dominant preference and uses it the most. MBTI states that this one dominant preference comes from the middle two inner preferences (sensing, intu-

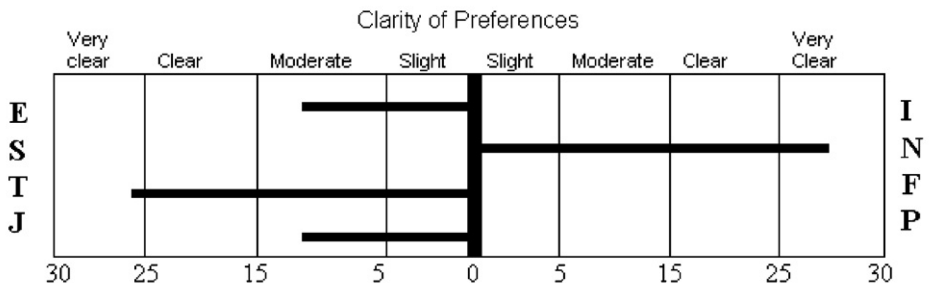

Fig. 1. The strengths of MBTI type preferences.

ition, thinking, and feeling), and not the two outer preferences [6,22,30]. A person uses the dominant type in his or her daily life; it is an essential part of the person at his or her best. Complementing the dominant type is the auxiliary or secondary type, which is called the second dominant type because it is used more than any other type aside from the dominant one. As with the dominant type, the auxiliary type is readily used and a person will often unconsciously shift back and forth between the dominant and auxiliary. This leads to the following possible combinations of ST, SF, NF, and NT (Fig. 2). The first letter of each type represents the dominant type and the second letter represents the auxiliary type. For example, ST can be found in E<u>ST</u>J, I<u>ST</u>J, E<u>ST</u>P, and I<u>ST</u>P. Extroverts (E) use their dominant preference mostly for the external world and introverts (I) use their dominant preference mostly for the inner world.

The prominent question in many studies that use MBTI is whether MBTI, being an independent variable, causes any significant result, or more specifically, whether a particular MBTI preference significantly influences the quality and productivity of the end-product. Cheng et al. [10] studied the business decision quality variation on various cognitive styles (sensing and intuition preferences of MBTI) of a group. A subject pool of 271 undergraduates was divided into three groups: homogeneous sensing preference pairs, homogenous intuition preference pairs, and diverse pairs with one of each preference. A production scheduling problem was given as a task, and the dependent variable, performance, was measured in terms of the average output of the production system for up to 15 periods completed by each pair. In a controlled laboratory environment, the subjects were given a 30 min training period, then in the main experiment they were allowed to perform up to 15 consecutive production periods or to continue the task for a maximum of 75 min. After each period, the subjects received their performance report. The results showed that cognitively diverse pairs performed significantly better than homogeneous sensing type pairs, but not better than intuition type pairs.

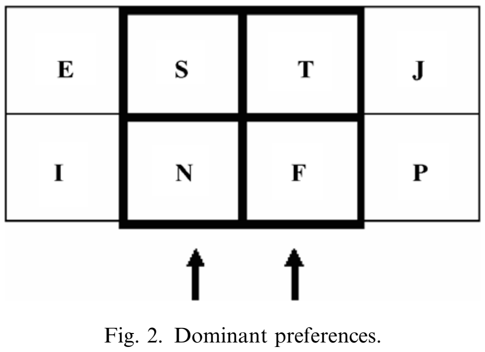

Fig. 2. Dominant preferences.

Bradley and Hebert [8] studied the productivity difference between two teams where each team included different personality types. A number of professionals from a medium-sized software development company were formed into two groups. The two teams were comparable in terms of age, intelligence, problem-solving ability, and task responsibility. Each team was given similar information systems development projects with comparable complexity. At the end, team one took almost twice as long as team two in delivering a finished information system. However, even though team one spent more time on developing the system, the user department was not satisfied with the system's quality. Moreover, team two produced a high quality level. After a thorough review and analysis of the teams' similarities and differences, one conspicuous difference was that of personality type difference. Team one had 80% introverts and 20% extroverts compared to team two's equal distribution of both types. The percentages of sensing type versus intuition type were comparable, with team one having 60% intuition type and 40% sensing type, while team two had 57% intuition type and 42% sensing type. A major difference in the thinking type versus feeling type makeup of the two groups was that team one had only 20% feeling type while team two had 42% feeling type. Team one had a better balance of judging type and perceiving type (70% J, 30% P) than team two (100% J). Based on these results, the study proposed that IS development team performance is at least partially related to the team's personality-type composition.

Are certain types better at programming? Weinberg [39] asserted that since different tasks of software development require different abilities there may be no one personality type which is best overall. However, certain personality types are better than others on certain aspects of software development. Da Cunha and Greathead [16] questioned whether a particular personality type tends to perform better at code review. A total of 64 undergraduates participated in the experiment. Each subject, working alone, was given 1.5 h to locate and identify as many bugs as possible in four pages of java code (282 code lines). There were 16 semantic bugs that were inserted earlier into the code. The results showed that the subjects with MBTI NT type achieved the highest scores. Students with NF, ST, and SF type performed the worst. Based on these results, it appears that students with N type tend to perform better than other types in code review.

Katira et al. [25] studied the pair programming experience and compatibility of under-represented groups, specifically female and minority students, in computer science. A very substantial number of subjects, including 1053 undergraduate and 112 graduate students, were involved. Three large groups were formed from three different classes (Introduction to Programming, Software Engineering, and Object-Oriented programming). The following attributes and data were adopted and used for the pairing process and analysis: MBTI, actual skill (measured by the multiple measures of midterm grade, overall GPA, and SAT/GRE score), 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. The results revealed that students are more compatible with partners whose programming skill levels are similar. Pairing two female students was likely to result in a compatible pair in the undergraduate classroom, while mixing gender pairs was less likely to lead to compatibility.

The results of the aforementioned studies and other supporting studies [9,19,26,31,38] infer that there is an underlying association between our decision-making and problem-solving processes. Furthermore, the heterogeneity of personality variables results in higher levels of team performance and quality in the end-product. This phenomenon would be more noticeable in a task-like programming because programming reveals an individual's personal views and understanding of a given problem [15,39]. The programmer's personality and tendencies – where a person focuses his or her attention, the way a person gathers information, the way a person makes decisions, and how a person deals with the problem – do contribute and manifest to a certain degree in the final end-product.

3. Experiment

3.1. Hypothesis development

The alike or homogeneous group may yield high likeability [25], but the diversely paired or heterogeneous group will outperform the homogeneous group [8,10,31]. As Isabel Briggs–Myers, one of the creators of MBTI, has observed, "two people, alike in their kind of perception or their kind of judgment but not both, have the makings of a good working relationship. Their shared preference gives them common ground and their dissimilar preference gives them, as a team, a wider range of expertness [emphasis added] than either has alone" [27]. Here, "expertness" also refers to insight or inquisitiveness. The two programmers, benefiting from the complementary pairing of perceptions, would yield the most optimal product. Moreover, based on the fact that a person feels most comfortable and is at his or her best while shifting back and forth between the dominant type and auxiliary type, we may theorize that the primary and auxiliary types support the intensive cognitive process needed for programming. Based on these facts, the following groupings are possible:

"Diverse" group: Pairs of subjects who are alike in EITHER their dominant OR auxiliary preferences but not both (ST–SF, NT–NF, ST–NT, and SF–NF). Labeled as [divrs] in this experiment.

"Alike" group: Pairs of subjects who are alike in BOTH their dominant AND auxiliary preferences (ST–ST, NF–NF, NT–NT, and SF–SF). Labeled as [alike] in this experiment.

"Opposite" group: Pairs of subjects who are completely OPPOSITE in BOTH their dominant and auxiliary preferences (ST–NF and NT–SF). Labeled as [opp] in this experiment.

H_1: [divrs] would score significantly higher than [**alike**] in code productivity.
H_2: [divrs] would score significantly higher than [**opp**] in code productivity.
H_3: [divrs] would score significantly higher than [**alike**] in code quality.
H_4: [divrs] would score significantly higher than [**opp**] in code quality.

3.2. Experiment execution

Initially, a group of professional programmers were sought to participate in the experiment. After a few unsuccessful attempts, however, the search shifted to university students. A few classroom visits were made to give presentations on XP, pair programming, and the experiment background, as well as the conditions for participating. The students and instructors were advised that each subject must consent and sign off on an agreement to allow the experiment facilitator full access to the subject's current course performance and GPA. Also, only subjects who finished the experiment would receive extra credit, the amount of which would be determined by the course instructor. However, no penalty was to be imposed on subjects who quit before the experiment ended.

A total of 128 students majoring in various computing majors started the experiment. Of the 68 undergraduates, 40 were first-year students and 28 were juniors; the remaining 60 were master's degree graduate students. Majors included Management Information Systems, Information Systems, and Information Technology.

Java and C++ were the programming languages being used by students in their various classes: one undergraduate C++ course, one undergraduate Java course, one graduate C++ course, and one graduate Java course. As these were introductory programming courses, the course objectives were to introduce basic programming concepts and algorithms rather than the specifics of C++ or Java. Hence treating both C++ and Java as similar languages for this experiment's purpose is appropriate.

The programming environment is provided in Table 1.

The subjects were required to make a total of three laboratory visits to complete the experiment. The main purpose of the first visit was to introduce the students to XP and pair programming, so there were only a few tasks. The students completed an online MBTI profile assessment, a pair programming training session, and filled out a questionnaire about programming experience (Appendix A). In the pair programming training session, any two subjects were paired because the objective was to acclimate the students to the pair programming environment. The experiment facilitator guided and coached the subjects on the key points of pair programming.

The distribution of the MBTI profile assessment reveals that the most common type was ISTJ with 21.1%, followed by ESTJ with 14.8% (Table 2). The least common type is INFJ with less than 1% (one student). From the two most common MBTI types, we cautiously project that both S preference and T preference contribute, to a certain degree, to an individual's interest in computing. Contrariwise, N and F preferences from the least common MBTI type may have minimum impact. This result is very similar to that of some previous studies [9,36].

After the first visit, a review of each subject's GPA, current course performance, past programming experience, pair programming experience, past programming courses, and the MBTI assessment result was compiled. One surprising fact learned from the questionnaires (Appendix A) was that many graduate subjects had neither prior programming experience nor had taken a college level programming course before. For the undergraduates, the class they were currently taking was their first programming course. None of the subjects had any professional programming experience or practice with extreme or pair programming. Given this situation, we categorized the subjects as having no prior programming experience.

Table 1
Programming environment

Categories	Specifications	Comments
Location		
Learning Systems Laboratory, GITC Building	11 PCs on a U-shaped table	Restricted area (accessible only by authorized personnel)
Hardware		
Personal computer	Dell PC Pentium III 1.6 GHz/512 MB/17″ monitor	None
Software		
Operating System	Microsoft Windows 2000 Professional	None
Programming Languages[a]	JAVA™ 2 Software Development Kit (J2SDK), Standard Edition, Version 1.4.2_03	Downloaded from Java home page http://java.sun.com/j2se/1.4.2/download.html
	C++ Microsoft Visual Studio 6	Installed by University Computing Systems Services

[a] No special editor software application used.

Table 2
MBTI distribution of the subjects

Sensing types		Intuitive types		
with thinking	with feeling	with feeling	with thinking	
Introverts				
Judging	ISTJ *N* = 27 21.1%	ISFJ *N* = 7 5.5%	INFJ *N* = 1 0.8%	INTJ *N* = 5 3.9%
Perceptive	ISTP *N* = 4 3.1%	ISFP *N* = 8 6.3%	INFP *N* = 9 7.0%	INTP *N* = 9 7.0%
Extroverts				
Perceptive	ESTP 9i = 7 5.5%	ESFP *N* = 7 5.5%	ENFP *N* = 7 5.5%	ENTP *N* = 4 3.1%
Judging	ESTJ *N* = 19 14.8%	ESFJ *N* = 7 5.5%	ENFJ *N* = 3 2.3%	ENTJ *N* = 4 3.1%

3.3. Pairing process

The next step was to pair the subjects, keeping in mind the necessity of filtering out any possible compounding variables such as course and grade level, and to manipulate the three MBTI type groups. Keeping the subjects in their respective courses, the subjects were placed into four groups (A, B, C, and Y) according to each student's performance level in the course. This was done to prevent the partner with the higher level of academic performance from doing most of the work in the experiment task.

The pairing process was done in a reasonable manner. In group A, a student with an "A" grade was paired with a student with either an "A" or "B+" grade. In group B, a student with a "B" grade was paired with a student with a "B," "B+," or "C+" grade. Group C consisted of students with a "C" grade and group Y included students with below "C" grades. Three different MBTI type groups of [divrs], [opp], and [alike] were formed within each of the four groups.

Since the subjects were categorized as having no prior programming experience, as explained previously, this factor was not considered for the pairing process. In summary, the subjects were paired primarily based on their MBTI types with regards to other aforementioned factors.

The subjects had no prior working or personal relationship before this experiment.

Considering the subjects' programming backgrounds and skills, a set of four programming problems (Appendix B) were designed for the experiment tasks. To validate the set for the difficulty parity level and the appropriateness of each problem, three outside professional programmers were used. They were briefed on the experiment and the programming backgrounds and skills of the subjects. The objective was for each problem to exhibit an equal level of programming difficulty. The set was validated by the programmers after careful evaluation and modification.

In the second visit, the subjects participated in a pair programming session and an individual programming session, both 45 min long. Two of the four programming problems were given, one per session, in this visit. The individual programming session was held so that the subjects could better differentiate and contrast pair programming from conventional individual programming, and to amplify the subjects' pair programming experience. The comparison between individual programming to pair programming is not addressed here as there are a number of existing studies [14,17,28].

The third visit was a repeat of the second visit, with one difference. If a subject had participated in the pair programming session first and the individual programming second, then the order was reversed in the third visit. Again, the alternation was done to amplify the PP experience. The other two programming problems were given in this visit, again one per session. The time limit of 45 min per session was strictly enforced, regardless of whether the work was completed or not. After the programming sessions, the subjects were debriefed on the pairing process and the expected hypotheses. Finally, the subjects were thanked for their participation and the experiment ended.

3.4. Independent and dependent variables

The pair types [divrs], [opp], and [alike] are the sole independent variables. The programming languages (C++ and Java) are not considered as variables because the students had no prior programming experience. Also, the students' introductory courses focus more on introducing algorithms and basic programming concepts, not the specifics of each language.

The dependent variable, or the variable programming output, is further broken down into two categories of code quality and code productivity. Quality refers to the number of programming defects or errors which cause the program to behave unpredictably or stop abruptly. Thus the fewer number of errors a program has, the higher the level of programming quality. However, this criterion would apply only to finished programs. Incomplete programs would only yield a number of errors on the incomplete work; hence the number of errors is not an accurate account of code quality. For example, a subject's completed program yields five errors while another subject's partially completed program yields only two errors. Are we to say the program with only two errors exhibits higher quality than the one with five errors? It is imperative that all subjects complete the given task prior to any formal code quality evaluation. If the work was not completed then the code quality measurement was not measured. In our experiment, many subjects with no prior programming experience had trouble completing the given programming task in time. Some subjects complained about the time shortage and some complained about their inability to finish the given programming task. Unfortunately, there were complications in arranging longer programming sessions. This is explained further in the discussion section.

With these shortcomings, the code efficiency is very difficult to measure. To be consistent and to respect the code efficiency definition, the code efficiency measurement is withdrawn for this experiment. This decision is also discussed further in the Section 5.

Similar to the code efficiency measurement, the code productivity measurement poses a similar challenge for objective definition and measurement. A common assumption is that a higher number of code lines indicates higher productivity. However, this approach requires justification and validation of the definition of "lines of code" [24]. One may argue that the lines of code consist of every line in the program including comment lines and data declaration lines. Another view is that only the executable lines should be counted for the measurement of productivity. Adding to this argument is the fact that high-level programming languages complete programming tasks with fewer lines of code than low-level programming languages. Whatever the case, the most suitable resolution should be decided case by case, In other words, code productivity can only be defined by giving consideration to specific circumstances and parameters such as programming language, programming platform, programming convention, programming task type, and the programmer's background. Under the scope of this experiment, code productivity can be defined as the combination of executable code and explicit, measurable coding effort. The coding effort can be any written code statement that contributes to the clarity or to the completion of the problem.

Complying with this definition of code productivity, two independent judges evaluated the subjects' codes. A Likert scale of zero to ten was used. A perfect work with the task coded completely, compiled with no errors, and providing accurate output received a score of 10. Work with no executable code, very limited coding effort, erroneous compilation, and no output received a score of zero. Scores for work that fell between these levels were given accordingly.

3.5. Other experiment parameters

To select the appropriate judges for the code productivity measurement, the credentials and programming experience backgrounds of two professional programmers were evaluated and reviewed by a panel of Information Systems faculty members. As with the subjects, the judges were also informed about the experiment and pair programming. However, the judges were completely blinded from the experiment hypotheses and the identity of the owner or owners of each programming work. Additionally, the judges worked independent of each other.

The time consumed variable refers to the time it took for the subjects to complete the given problem. The 45 min timeframe was strictly enforced, even if the subjects had not completed the work.

4. Results

Table 3 shows the number of subjects and sample size, while Table 4 shows the mean scores and standard deviation values for code productivity. The "Subjects" column indicates the total number of participating subjects per group. The total number of subjects from all three groups was reduced from 128 to 124 because the four subjects (one pair of [divrs] and one pair of [alike]), withdrew from the experiment due to personal reasons. The "Pairs" column indicates the number of pairs in each group. The data points, under "N," represent the number of total pair programming sessions for each group. For example, the two pair programming sessions by 20 [opp] pairs yielded a total 40 data points, or one data point per session. For [alike], the data points are 43 instead of 44 because one pair of pair programming session data was inadvertently lost. Lastly, mean scores and the standard deviation values are calculated. [divrs] has the highest mean score (6.46) and the lowest standard deviation value (2.43). It is followed by [opp] (mean score 5.31, SD 3.19), and [alike] (mean score 4.64, SD 3.03).

Next, a one-way analysis of variance (ANOVA) test is needed for the significance test on code productivity values. Before beginning the test however, it is imperative to determine whether the data qualified for the parametric test [18,32]. ANOVA is only to be used with normally distributed data. The four criteria of normally distributed data, homogeneity of variance, interval data, and independence must be checked for the data to qualify [18,32,33]. The normal distribution refers to the bell-shaped distribution of data. Any skewed shaped data will not qualify for the parametric test. Homogeneity of variance means that the variances should not change systematically throughout the data. Interval data means that data should be measured at least at the interval level. Independence means that data from different subjects are independent. In other words, the behavior of one participant does not influence the behavior of another.

In determining if our data is normally distributed data or not, a histogram (Fig. 3) and a *Q*–*Q* plot (Fig. 4) are drawn.

However, the visual examination of the data provided by the histogram and plot does not clearly accredit the data as normally distributed. Striving for a more accurate analysis, a Kolmogorov–Smirnov (K–S) test is used to compare the sample data to normally distributed data with the same mean and standard deviation. If the K–S test result is nonsignificant ($p > 0.05$) then the sample data is normally distributed. However, the K–S test result showed the signif-

Table 3
Descriptive of [pairtype]

	*N**	Pairs	Subjects
[opp]	40	20	40
[divrs]	40	20	40
[alike]	43	22	44
Total	123	62	124

*N** = the number of total pp sessions for each [pairtype].

Table 4
Code productivity score analysis

	[opp]	[divrs]	[alike]
Code productivity mean score	5.31	6.46	4.64
Code productivity score std. dev.	3.19	2.43	3.03

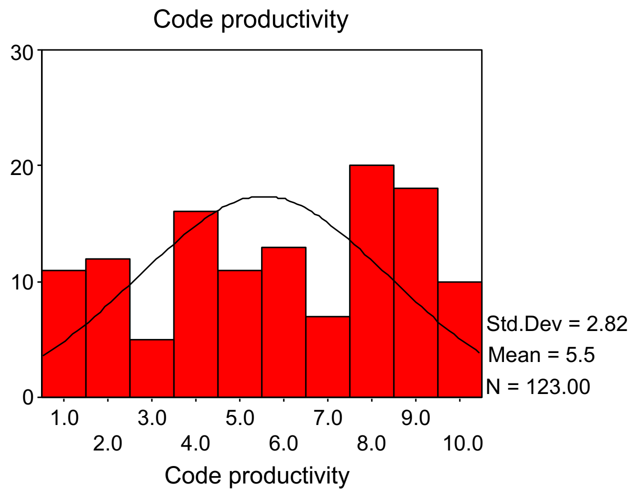

Fig. 3. Code productivity histogram.

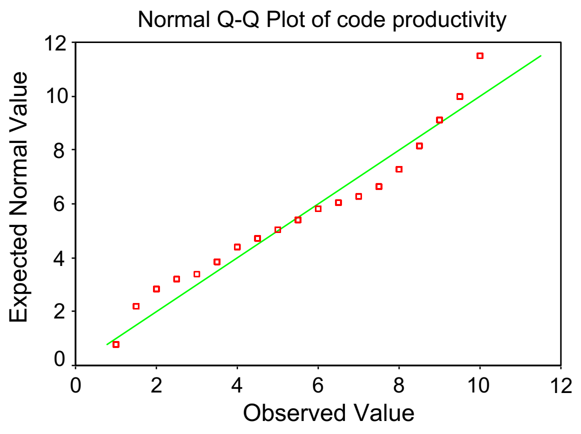

Fig. 4. Code productivity *Q*–*Q* graph.

Table 5
Kolmogorov–Smirnov (K–S) test

Statistic	df	Sig.
0.145	123	0.000

icant value as 0.000 (Table 5), which means that our data does not qualify as normally distributed.

Due to this disqualification, various means of data transformation were tried: logarithmic transformation, square root transformation, reciprocal transformation, exponential transformation, power transformation, and arcsine transformation. However, the results were not normally distributed. These failures limited us to using nonparametric tests. Nonparametric tests for multiple independent samples are useful for determining whether or not the values of a particular variable differ between two or more groups.

The most appropriate nonparametric test for this situation appeared to be the Kruskal–Wallis test. It is the nonparametric analogy of a one-way analysis of variance (ANOVA) and compares the medians of two or more samples ([divrs], [opp], and [alike] in this experiment) to determine if the samples come from different populations. The test also assumes that there is no a priori ordering of the k populations from which the samples are drawn. Table 6 shows the Kruskal–Wallis test result.

The Kruskal–Wallis test uses ranks of the original values and not the values themselves. Each score or data point is ranked without regard to group membership. Scores tied to a particular value receive the average rank for that value. After ranking the scores, the ranks are averaged within groups. The Kruskal–Wallis statistic measures how much the group ranks differ from the average rank of all groups. The degrees of freedom (df) for the chi-square statistic are equal to the number of groups minus one. Table 7 shows the χ^2 test result. We have three groups hence the df value is two. The asymptotic significance estimates the probability of obtaining a chi-square statistic greater than or equal to 7.067, if there truly are no differences between the group ranks. A chi-square of 7.067 with two degrees of freedom should occur only about 29 times per 1000. We may say that the three groups are

Table 6
Kruskal–Wallis test

	[opp]	[divrs]	[alike]
N	40	40	43
Mean rank	60.95	73.22	52.53

Table 7
Chi-square

Chi-square	df	Asymp. Sig.
7.067	2	0.029

Table 8
Mann–Whitney test I

	[opp] vs. [divrs]		[opp] vs. [alike]		[divrs] vs. [alike]	
N	40	40	40	43	40	43
Mean rank	36.76	44.24	44.69	39.50	49.49	35.03
Sum of ranks	1470.50	1769.50	1787.50	1698.50	1979.50	1506.50

Table 9
Mann–Whitney test II

	[opp] vs. [divrs]	[opp] vs. [alike]	[divrs] vs. [alike]
Mann–Whitney *U*	650.500	752.500	560.500
Wilcoxon *W*	1470.500	1698.500	1506.500
Z	−1.443	−0.983	−2.737
Asymp. Sig. (two-tailed)	0.149	0.326	0.006
Exact Sig. (one-tailed)	0.075	0.163	0.003

significantly different if the asymptotic significance is 0.05 or below. Table 6 shows that the code productivity significantly differed by pairtype. Like the F-test in standard ANOVA, Kruskal–Wallis does not tell us how the groups differed, only that they are different in some way.

To check for significant differences between groups, we used the Mann–Whitney test, which is another nonparametric test. As shown in Tables 8 and 9, the Mann–Whitney test totaled three sets: [opp] vs. [divrs], [opp] vs. [alike], and [divrs] vs. [alike]. Before a brief explanation of the results, one must remember that the most important statistical value from Table 9 is the exact significant value. As in the Kruskal–Wallis test, the data points, or scores, from the two groups are combined and ranked from lowest (1) to highest (N). Each data point or score is ranked without regard to group membership. Scores tied to a particular value receive the average rank for that value. After ranking the scores, the ranks are averaged and summed within the groups. The Mann–Whitney *U* for either group is equal to the difference between the maximum possible value of summed rank for the sample, versus the actually observed value of summed rank. The Mann–Whitney *U* value displayed in Table 9 is the smaller of these two values. The Wilcoxon W value is simply the smaller of the two rank sums from each set. Again, the most important statistical value from Table 9 is the exact significant value. The asymptotic significance value is based on the assumption of a normally distributed data set. The exact significance value is when the data set is not normally distributed, which is the case here. Using a significance level of 0.10 instead of 0.05, [divrs] shows significant differences from both [opp] and [alike] ($p = 0.075$, $p = 0.003$). No significant difference is shown between [opp] and [alike] ($p = 0.163$).

These results support two of the four hypotheses, that [divrs] would score significantly higher than [alike] in code productivity, and that [divrs] would score significantly higher than [opp] in code productivity.

Out of 123 PP sessions, only 15 finished the task before the 45 min allowed time and most of the subjects consumed the maximum time. In the second visit (first PP session), only one pair from each group completed the task in less than 45 min: [divrs] finished at 32 min, [opp] at 40 min, and [alike] at 39 min. In the third visit (second PP session), there were a total of 12 pairs completing the task in less than 45 min [divrs] led with five pairs, [opp] with four pairs, and [alike] with three pairs. This is only 12% of the total data points (15/123 = 12.2%). However, this low completion is not a problem for this experiment's objective. Because the value is not on whether the subject has completed coding the assigned programming problem but rather on how much the subject has made progress on coding in the given time period. The two independent judges evaluate the subject's coding effort on a Likert scale of zero to ten, regardless of whether the subject finished coding or not.

5. Discussion

Unlike many previous empirical studies involving MBTI [8,10], this study features the synchronous involvement of the dominant preference, or the two middle inner preferences of sensing, intuition, thinking, and feeling, and the phenomenon of the frequent, unconscious shift between the dominant and auxiliary preferences. This study addresses the relationship between preferences and the programming productivity and efficiency of pair programming. Despite the unexpected limited results, it does appear based on the productivity result that the personality variable has a noticeable impact on code productivity within the team programming context. More specifically, the current work suggests that if two people, with no programming experience, who are alike in either their MBTI dominant or auxiliary preferences but not both (ST–SF, NT–NF, ST–NT, and SF–NF), are paired together, their productivity level will be significantly higher than that of any other MBTI pair combination.

The diversity and complementary nature of the diverse pair provides unanticipated angles as well as a more comprehensive view in the pair's problem solving approach. In addition, the "checks and balances" system provided by differing personalities help keep the two people on track, while their similarities provide common ground for reconciling differences. In an active pair programming session, this implicit cognitive collaboration would serve as the driver in completing the task. For example, let us consider a [divrs] pair with an ST–NT makeup. One member is partial to the sensing preference, which is attuned to practical solutions and concrete information. The other member is partial to the intuition preference, which is attuned to theoretical implications or new possibilities of events. These two preferences are opposite to each other, and can thus provide diverse problem solving approaches. While both members are different with regards to their sensing and intuition preferences, they are similar in their thinking preferences. While the sensing and intuition differences create diversity, the similarity in thinking preference gives the team common ground and a greater sense of compatibility. In sum, the pair's common thinking preference would result in compromise and concession, and each member's diverse sensing and intuition preferences would challenge each other and offer a wider array of ideas and solutions.

The results from the other two groups, [opp] and [alike], draw an interesting picture. The [opp] group displays a higher level of productivity than the [alike] group. In other words, pairs of subjects who are completely opposite in both their MBTI dominant and auxiliary preferences (ST–NF and NT–SF) outperformed the pairs of subjects who are alike in both their MBTI dominant and auxiliary preferences (ST–ST, NF–NF, NT–NT, and SF–SF). A conservative explanation of this phenomenon is that in [opp] pairs, a system of "checks and balances" exist, which ultimately yields a better end-product, but there are also conflicts which compromise the productivity to some degree. With the [alike] groups, the system of "checks and balances" that is present in the dynamics of the [divrs] and [opp] groups is missing, which may result in lower levels of productivity.

5.1. Limitations

The most profound limitation is the measurement of code productivity. Additionally, the question of how one defines code productivity is significant. What constitutes code productivity in a given context and how can it be measured accurately and completely? This intricate issue is compounded by the subjective evaluation of human judges, which is questionable because it can be biased. The obvious risk is that one may receive different evaluations of the same work from different judges.

Another serious limitation is the missing code quality measurement. This is mainly due to the incomplete coding work of the subjects. The reason why many subjects did not finish the coding work can be attributed to the time limit, the difficulty level of the problems, or unknown variables. However, if the code quality measurement was available, then the correlation between code productivity and code quality could be analyzed, using multivariate analysis of variance (MANOVA), for any subtle relationships between code productivity and quality. A logistical limitation is the short duration of each pair programming session. One training session, two individual programming sessions of 45 min each, and two pair programming sessions at 45 min each are not sufficient to truly experience real-life programming. However, given the present university research environment, what is considered to be legitimate? Three hours? A semester? What time period is appropriate?

In using student subjects, the limitations can be their novice-level programming ability and newness to pair programming protocols. Consequently this arrangement may fall short of meeting the experiment's objective. Knowing this possibility, attempts were made in the beginning to solicit professional pair programmers' participation. However, the attempts were very challenging and unsuccessful. Using student subjects is not an ideal decision, but unfortunately in many academic experiments, it is the only option available.

Using students poses another difficulty, especially in code efficiency. This aspect of the experiment was challenged before the experiment. There is a conflict between the course instructor's objective and the experiment's objective. Obtaining the course instructor's consent to execute a semester-long pair programming session would be one of the preliminary experiment design action items. Using experienced professional programmers with at least a few years of experience in pair programming or team programming, and running the experiment for several months would have been ideal for this experiment. The pool of professional programmers may have yielded more insights and more underlying subtleties on psychosocial factors and, more importantly, their magnitudes.

Lastly, using only one measuring tool, MBTI, and generalizing the personality impact assessment on collaborative programming are an injustice to all other valid personality assessment tools.

5.2. Recommendations

Given the limitations, another run of this experiment with much more rigorous parameters is deemed appropriate. Instead of a 45 min programming session, there should be no time limit. Since the major deficiency of this experiment largely stems from the short programming sessions, the open session arrangement would not only contribute to the code quality measurement, but it would also yield a more desirable overall result. If an open session arrangement is not administratively feasible then more sessions could be scheduled.

The second parameter is the choice of subject pool. Using student surrogates for professional programmers exposes the empirical result to questions and challenges. Programming professionals or even students with programming experience would strengthen the results. The challenge lies in allocating a substantial number of subjects to achieve statistically acceptable data points.

Furthermore, diversifying some of the parameters to other choices may induce some intriguing interactions. For example, replacing MBTI with other equally competent personality profile instruments can be used to validate or even challenge the findings of this experiment. Other instruments would size or survey personality differently. Hence, a different preference combination or pair would surface. Expanding the number of programmers to three or more per team would certainly lead to different layers of collaborative team dynamics.

6. Conclusions

The current study demonstrates a clear indication of the difference in PP code productivity among different MBTI pair types. Although MBTI and other personal psychosocial traits are hidden and time-consuming to identify, underlying personal attributes do assert a certain degree of influence on overall production in the team programming environment. However, this study's results should be carefully interpreted as MBTI is only one of many popular personality assessment instruments, and the experiment design can be altered in many different ways. Acknowledging the fact that programming is only a part of what makes up a software project, the value of [divrs] group is yet to be tested. For the same reasons, [alike] and [opp] groups must not be counted out just because of this study's results.

Understanding and respecting this psychosocial phenomenon of team programming by management will greatly enhance efficiency and maximize programming production. MBTI is one of the most common means of gathering new employee information and many large corporations retain this record in their Human Resources archives. A typical scenario in programming is that employees are allowed to team up among themselves with no or minimal interference or instruction from management. An employee often teams up with another employee only because they are friendly, not to maximize production. Even if there is interference or instruction from management, it is most likely not based on the psychosocial phenomenon of team programming. The application of this study's results, or other significant psychosocial phenomenon, does not require a great deal of preparation, resources, or a high price tag-like other technical techniques and software applications that claim to enhance overall programming experience or production; it is a valuable source that can be applied or administered easily for great effect.

Appendix A

Questionnaire used to collect the programming courses and backgrounds of subjects.

Programming experience information	
Name:	Student ID:
Major:	School year:
Your current programming course:	
List all programming courses that you have taken (college level)	
List all your professional programming experiences (Jobs, How long)	
Have you practiced extreme programming before? If so, where and how long?	
Have you practiced pair programming before? If so, where and how long?	

Appendix B

The four programming tasks given to the subjects

Problem: employee pay

Write a program called "Pay" which calculates the pay for an employee. The program let the user enter the following information: name, status (full/part-time), number of hours worked, and job payrate. Your program should output the employee's name, the number of hours worked and total pay.

- If an employee worked 40 h or less, then the employee should get the job (regular) payrate.
- If an employee worked between 40 and 60 h (from 41 to 60) and the employee is a full time employee, then the employee gets the double overtime payrate (job payrate × 2), but if the employee is a part time employee then the employee gets time and a half (job payrate × 1.5).
- If an employee worked between 60 and 80 h (from 61 to 80) and the employee is a fulltime employee then the employee gets triple overtime (job payrate × 3), but if the employee is a part-time employee then the employee gets double overtime.
- If an employee works more than 80 h, nothing is calculated and your program should print an error message that states, "meet with your management".

Problem: life insurance

An insurance company offers three plans: Plan A at $100,000, Plan B at $500,000, and Plan C at $ 1,000,000. For Plan A the standard monthly premium (stdrt) is $10, plan B is $20, and plan C is $50.

Write a program that calculates the monthly premium. The user inputs name, age, gender, medical history, marital status, smoking/non-smoking, exercise/no-exercise, years with the plan, availability to change options and whether it is new or a renewal. The program outputs the customer's name, age, gender, and the recalculated monthly premium.

If the customer has been with the plan for 5 or more years the customer has the option to change to a different plan every 5 years. However, if the customer wants to change when he or she has had the plan for less than 5 years, or wants to change on a year that is not a fifth year, he or she may do so with a penalty of 1.05 factor.

Plan B is the most profitable product. If an organization with 150 employees or more chooses Plan B for all their employees, then each employee receives a 0.95 factor.

The factors are cumulative, so if a customer is 37 years old, male, a medical "high" risk, married, a non-smoker, exercises regularly and changing the plan with a penalty, then the recalculated premium would be (1.2 * 1.0 * 1.5 * 0.8 * 0.7 * 1.0 * 1.05) * stdrt. For a condition that is not listed in the table use 1.0 (i.e. male, non-smoker).

Condition	Factor
Age < 36	0.7
35 < age < 48	1.2
47 < age < 65	1.5
64 < age	1.7
Female	0.9
Medical history "high" risk	1.5
Medical history normal condition	1.0
Medical history "low" risk	0.8
Married	0.8
Single	1.2
Smoker	1.5
Regular exercising	0.7

Problem: online supermarket

Your team is asked to create a program that is to be used by an online supermarket. You need to observe the following;

1. Each item in the store has a standard price that is up to two decimal places in precision.
2. A state sales tax (7%) is charged on all items except for bottled water, toilet paper, and fresh fruit.
3. A customer must buy at least 5 items, which includes 2 or 3 promo items and onenon-taxed item.
4. A customer may buy more than one quantity per item.
5. The extended price (unit price * qty) for an item should never be less then zero.
6. The store may run a promotion where the customer can buy three items for a special price. If the customer buys less than three while the promotion is in effect, the price is equal to the special price divided by the quantity rounded down to two decimal places.
7. Rounded down to two decimal places.

Items	Price ($)	Promotions?	Special Price ($)
Loaf of bread	1.99		
Carton of orange juice	5.99		
One pound of apples	2.99		
Box of ice cream	3.99		
Set of two rolls of paper towels	1.99	Yes	3 for $4.99
Bottled water	0.99		
Bottle of ketchup	4.99		
Pecan pie	3.99		
Box of laundry detergent	8.99	Yes	3 for $14.99
Bottle of olive oil	2.99		
Bag of grapes	4.99		
Set of 12 rolls of toilet paper	9.99	Yes	3 for $16.99
Case of tofu	1.99		
Bag of imported black beans	2.99		

Input is where a shopper shops online (selects the items) and the output is a list of items with the price of each item and the total balance.

Problem: emergency medic

During a time of war, field surgeons and medics are always in shortage. As a remedy for this problem, the Department of Defense is testing an idea where junior medic soldiers can be deployed to give medical treatment. To help the junior medic soldier, a PDA containing special program is to be used. It is to give minimal but life-saving emergency treatment. With the help of the PDA, the medic soldier assesses the wounded soldier and performs appropriate treatment. As the lead developer, you are asked to create the program. For this pair programming experiment purpose, please use your common sense regarding the appropriate treatments.

The program asks the following questions and depending on the answers a treatment plan will be the output.

- Types of wounds
  - abdominal wound
  - head wound
  - shattered bone
  - broken bone
  - burn injury
  - injury from explosive devices fragments
  - bio-chemical injury.

- Pulse rate
- Body temperature
- Conscious or unconscious
- Severe bleeding or not
- Experiencing difficulty breathing

Output will be the soldier's name, soldier's military ID, list of all inputted entries, and the list of appropriate treatments. Treatments

- Remove large/noticeable fragment pieces.
- Tightly wrap the wounded area with a roll of gauze.
- Apply antibiotic cream.
- Protect and shield the wounded area with a flexi-support beam.
- Wash wounded area.
- Perform a pain-killing shot.
- Perform an antibiotic shot.
- Place a medic band-aid.
- Immediate transport to a field hospital required.
- Elevate legs.

As a medical note, the following conditions apply:

(1) Any head wounds and unconsciousness require immediate transport to a field hospital as a final step.
(2) If the body temperature is above 105 °F or below 90 °F, do not perform the pain-killing shot.
(3) A shattered bone injury receives 1.5 times dosage of painkiller.
(4) Regular bleeding may get the medic band-aid, but severe bleeding does not.
(5) For a bio-chemical injury, do not wash with water on the area.

References

[1] A. Anderson, R. Bettie, K. Beck, Chrysler goes to extreme, Distributed Computing (1998) 24–28.

[2] D. Arnott, Cognitive biases and decision support systems development: a design science approach, Information Systems Journal 16 (1) (2006) 55–78.

[3] AgileManifesto, Avaliable from: http://www.agilemanifesto.org.

[4] R. Baskerville, J.P. Heje, EMethodology: towards a systems development methodology for e-business and e-commerce applications, in: S. Elliot, K.V. Anderen, P. Swatman, S. Reich (Eds.), Developing a Dynamic, Integrative, Mult-Disciplinary Research Agenda in E-Commerce/E-Business, BICE Press, Newcastle, Australia, 2001, pp. 145–159.

[5] R. Baskerville, J. Pries-Heje, Short cycle time systems development, Information Systems Journal 14 (3) (2004) 237–264.

[6] R. Bayne, The Myers–Briggs Type Indicator: A Critical Review and Practical Guide, first ed., Nelson Thornes Ltd., London, United Kingdom, 1995.

[7] K. Beck, Extreme programming explained: embrace change, Addison-Wesley, Reading, MA, 2000.

[8] J.H. Bradley, F.J. Hebert, The effect of personality type on team performance, Journal of Management Development 16 (5) (1997) 337–353.

[9] L.F. Capretz, Personality types in software engineering, International Journal of Human–Computer Studies 58 (2) (2003) 207–214.

[10] M.M. Cheng, P.F. Luckett, A. Schulz, The effects of cognitive style diversity on decision-making dyads: an empirical analysis in the context of a complex task, Behavioral Research in Accounting 15 (1) (2003) 39–62.

[11] K.S. Choi, F.P. Deek, Extreme programming, too extreme?, Proceedings of the International Conference on Software Engineering Research and Practice (SERP'02), Las Vegas, Nevada, USA., 2002, pp. 501–507.

[12] A. Cockburn, Agile Software Development, Addison-Wesley, Reading, MA., 2001.

[13] A. Cockburn, J. Highsmith, Agile software development, the people factor, Computer 34 (11) (2001) 131–133.

[14] A. Cockburn, L. Williams, The Costs and Benefits of Pair Programming in Extreme Programming Examined, Addison-Wesley, Reading, MA, 2001.

[15] L.L. Constantine, Constantine on Peopleware, Yourdon Press, Upper Saddle River, NJ, 1995.

[16] A.D. Da Cunha, D. Greathead, Code review and personality: is performance linked to MBTI type? Newcastle upon Tyne: University of Newcastle upon Tyne, Computing Science, 2004.

[17] H. Erdogmus, L. Williams, The economics of software development by pari programmers, The Engineering Economist 48 (4) (2003) 283–319.

[18] A. Field, Discovering Statistics: using SPSS for Windows, Sage Publication, Thousand Oaks, CA, 2003.

[19] W.L. Gardner, M.J. Martinko, Using the Myers–Briggs type indicator to study managers: a literature review and research agenda, Journal of Management 22 (1) (1996) 45–83.

[20] R.L. Glass, Extreme programming: the good, the bad, and the bottom line, IEEE Software 18 (6) (2001) 111–112.

[21] J. Grenning, Launching extreme programming at a process-intensive company, IEEE Software 18 (6) (2001) 27–33.
[22] S.K. Hirsh, J.M. Kummerow, Introduction to Type in Organizations, second ed., Consulting Psychologists Press, Palo Alto, CA, 1990.
[23] C.M. Jessup, Applying psychological type and gifts differing to organizational change, Journal of Organizational Change Management 15 (5) (2002) 502–511.
[24] T.C. Jones, Measuring programming quality and productivity, IBM Systems Journal 17 (1) (1978) 43–50.
[25] N. Katira, L. Williams, E. Wiebe, C. Miller, S. Balik, E. Gehringer, On understanding compatibility of student pair programmers. Proceedings of the 35th SIGCSE technical symposium on Computer science education SIGCSE'04. 2004, pp. 7–11.
[26] S.L. Kichuk, W.H. Wiesnera, The big five personality factors and team performance: implications for selecting successful product design teams, Journal of Engineering and Technology Management 14 (3) (1997) 195–221.
[27] I.B. Myers, P.B. Myers, Gifts Differing: Understanding Personality Type, Davies-Black, Palo Alto, CA, 1995.
[28] J.T. Nosek, The case for collaborative programming, Communications of the ACM 41 (3) (1998) 105–108.
[29] L.R. Offermann, R.K. Spiros, The science and practice of team development: improving the link, The Academy of Management Journal 44 (2) (2001) 376–392.
[30] N.L. Quenk, Beside Myself: The Inferior Function in Everyday Life, Consulting Psychologists Press, Palo Alto, CA, 1993.
[31] R.R. Reilly, G.S. Lynn, Z.H. Aronson, The role of personality in new product development team performance, Journal of Engineering and Technology Management 19 (1) (2002) 39–58.
[32] R. Rosenthal, R.L. Rosnow, Essentials of Behavioral Research: Methods and Data Analysis, second ed., McGraw-Hill, 1991.
[33] D. Rowntree, Statistics Without Tears: A Primer for Non-mathematicians, Pearson, Boston, MA, 2003.
[34] W.W. Royce, Managing the development of large software systems: concepts and techniques, Proceedings of the ninth International Conference on Software Engineering (1987) 328–338.
[35] J. Sample, The Myers–Briggs type indicator and OD: implication for practice from research, Organization Development Journal 22 (1) (2004) 67–75.
[36] G.D. Sterling, T.M. Brinthaupt, Faculty and Industry conceptions of successful computer programmers, Journal of Information Systems Education 14 (4) (2003) 417–424.
[37] D. Stotts, L. Williams, N. Nagappan, P. Baheti, D. Jen, A. Jackson, Virtual teaming: experiments and experiences with distributed pair programming, Lecture Notes in Computer Science (2003) 129–141.
[38] T. Varvel, S.G. Adams, S.J. Pridie, B.C. Ruiz Ulloa, Team effectiveness and individual Myers–Briggs personality dimensions, Journal of Management in Engineering 20 (4) (2004) 141–146.
[39] G.M. Weinberg, The Psychology of Computer Programming, Silver anniversary ed., Dorset House, New York, 1998.
[40] L. Williams, R.R. Kessler, W. Cunningham, R. Jeffries, Strengthening the case for pair programming, IEEE Software 17 (4) (2000) 19–25.
[41] L. Williams, R.R. Kessler, Pair programming illuminated, Addison-Wesley, Reading, MA, 2002.
[42] L. Williams, R. Upchurch, In: support of student pair programming. The 32nd Technical Symposium on Computer Science Education, Charlotte, NC, USA, 2001, pp. 327–331.