Journal of Information Systems Education

Volume 29
Issue 2
Spring 2018

Do Pair Programming Approaches Transcend Coding? Measuring Agile Attitudes in Diverse Information Systems Courses

Kuanchin Chen and Alan Rea

Recommended Citation: Chen, K. & Rea, A. (2018). Do Pair Programming Approaches Transcend Coding? Measuring Agile Attitudes in Diverse Information Systems Courses. *Journal of Information Systems Education*, 29(2), pp. 53-64.

Article Link: http://jise.org/Volume29/n2/JISEv29n2p53.html

Initial Submission:	16 August 2017
Accepted:	30 January 2018
Abstract Posted Online:	21 March 2018
Published:	13 June 2018

ISSN: 2574-3872 (Online) 1055-3096 (Print)

Do Pair Programming Approaches Transcend Coding? Measuring Agile Attitudes in Diverse Information Systems Courses

Kuanchin Chen
Alan Rea
Department of Business Information Systems
Haworth College of Business
Western Michigan University
Kalamazoo, MI 49008, USA
kc.chen@wmich.edu, alan.rea@wmich.edu

ABSTRACT

Agile methods and approaches such as eXtreme programming (XP) have become the norm for successful organizations not only in the software industry but also for businesses seeking to improve internal software processes. Pair programming in some form is touted as a major functionality and productivity improvement. However, numerous studies show that simply placing two programmers side by side in front of a single computer screen is not enough. We must look at other factors such as programmer expertise, project preparation, and perceived solution quality to understand pair programming's promises and pitfalls. In our study, we apply tailored programming challenges to a multifaceted group of first-year through senior Information Systems (IS) and non-IS majors to analyze how participant attitudes and perceived benefits of pair programming change from pre- to post-study, as well as determine whether the quality and functionality of the solutions differ across education levels and disciplines. Our findings show a strong interaction effect of gender and major composition (CIS vs. non-CIS majors) in all four dimensions of the ATMI attitude scale. Findings also suggest that experience in problem solving and solution formation are more important than prior specific domain knowledge. Finally, participants' perceived ability, sense of accomplishment, and completion of the assigned work, regardless of background or demographic, determined their performance outcome on the pair-programming tasks, which suggests that not all forms of attitude and perceived benefits contribute to the performance outcome.

Keywords: Pair programming, Agile, Extreme programming, Student attitudes, Attitudes towards mathematics inventory (ATMI), Productivity, Problem solving

1. INTRODUCTION

Agile methods have been accepted in many modern organizations. Well-established traditional (e.g., finance), as well as technology (e.g., software development), companies are using eXtreme programming (XP) approaches to keep pace with contemporary development cycles (Canfora et al., 2007; Vanhanen and Mäntylä, 2013). In this paper, we focus on one of the more common XP approaches, pair programming, and its effect on attitude changes, functionality, and solution quality. In its simplest form, pair programming has two programmers sitting side by side in front of one computer system. The driver sits at the keyboard and is responsible for inputting code, deciding on logic structures, etc. The navigator sits next to the driver and (to use a manufacturing metaphor) oversees production by watching for syntax errors. Moreover, the navigator makes sure the program meets client requirements and deliverables. After some time, the programmers switch roles and the process continues. Advocates of this method assert that paired programmers catch and address more errors, improve their programming approaches, produce better code due to collaborative cognitive efforts, and are more satisfied with the process (Flor, 1998; Nosek, 1998; Williams and Kessler, 2000; Williams, Wiebe, and Yang, 2002).

Many in academic and industry settings, as well as programmers in general, accept all, or some, of these premises as true. However, as with all methodologies, we find that this one – sitting two programmers together to perform one task – is not as straightforward as it may seem. Ever since collaborative programming (Nosek, 1998) was advanced 20 years ago as a software engineering method, researchers have examined whether these assertions hold true across the myriad of contexts in which pair programming is practiced. The focus of these studies include professional programmers (Bryant, Romero, and du Boulay, 2008; Nosek, 1998; Tingling and Saeed, 2007) and students – both novice and experienced – (Sanders, 2002; Williams and Kessler, 2001; Williams and Upchurch, 2001). Research has occurred in both controlled lab situations (Bryant,

Romero, and du Boulay, 2008; Cockburn and Williams, 2001; Domino, Collins, and Hevner, 2007; Lui and Chan, 2006) and within more open approaches (Sherrell and Robertson, 2006). All studies attempt to delineate constructs (Wray, 2010) that spread the resulting analysis across a broad range, from successful (Bryant, Romero, and du Boulay, 2008; Domino, Collins, and Hevner, 2007; McDowell et al., 2002; Nagappan et al., 2003; Williams et al., 2000), to mediocre (Dybå et al., 2007), to abysmal (Nawrocki et al., 2005; Stephens and Rosenberg, 2004).

Regardless of this considerable research, there is scant empirical evidence endorsing pair programming's use outside of software development notwithstanding the general consensus that two heads together are better than one (e.g., Nosek, 1998). Dybå et al. (2007) cautioned that the benefits of pair programming may be affected by a participant's experience and the task characteristics. Similarly, Koriat (2012) suggested that confidence, communication, and other factors were the true reasons for better results. Therefore, simply performing pair programming cannot be the only reason for its touted benefits. Dynamics among people, task, and methodology are keys to its success. From this perspective, this research is designed to accomplish the following objectives:

- To examine how participant attitudes and perceived benefits of pair programming are related to the quality of the solution, and
- To study whether the quality of the solution produced via pair-programming varies across multiple disciplines.

In Section 2, we examine already published research supporting our approach and our addition of attitude assessment as a viable method for measuring programming's technical nature. We then outline our methodology in Section 3, followed by a detailed findings analysis in Section 4. In Section 5, we discuss how our results are relevant to the existing debate, plus explain what worked and what did not concerning pair programming and attitude adjustments. Moreover, we theorize why these results concur with some historical research constructs but do not support others. After noting our study's contributions and limitations, we conclude our research in Section 6 and pose questions for further consideration.

2. LITERATURE REVIEW

Asking a colleague to help collaboratively solve a complex issue is nothing particularly innovative and occurs daily in various organizational contexts. However, redefining a process through which computer programmers can solve algorithm and coding challenges as a team rather than individually is still novel even though the concept was first expressed two decades ago (Nosek, 1998). Nosek's concept of "collaborative programming" (1998, p. 106) advocated that programmers working in two-person teams would produce more functional solutions in less time with greater satisfaction and confidence.

2.1 Collaborative Programming

In his study, Nosek (1998) paired 10 of 15 experienced programmers. The teams and the five individual programmers (as a control group) all were tasked with solving a database consistency check using the C programming language in 45 minutes. None previously had attempted this because the company had always outsourced it. Nosek found that, on average, independently-working programmers required at least 12 minutes more than pairs to complete the task. Pairs also reported enjoying the process more than individuals and, in some cases, produced better code than the company's outsourced, specialized consultants (Nosek, 1998, p. 107b). Although a strong case for collaborative programming was made, Nosek did observe that senior programmers' results exceeded novices' regardless of them working alone or in pairs.

2.2 Pairing Dyads Importance

Nosek's finding of programmer experience level suggests that collaboration might be beneficial only if pairings are expert-novice, rather than expert-expert; this parameter was integrated into our study via an experience question on a pre-survey that measured the participants' familiarity with the topical exercises. Other researchers have noted that partner pairing selection is critical for successful pair programming (Lui, Barnes, and Chan, 2010; Wray, 2010) and have proposed pairing experts with novices for maximum educational benefits (Domino, Collins, and Hevner, 2007, p. 305b).

2.3 Pair Programming Tenets

Williams and Kessler (2000) solidified many of the tenets of what has become pair programming, such as share everything, avoid preconceptions, and focus on the tasks. For many implementing pair programming studies, reading Williams and Kessler's article is a prerequisite for study participants. Although we did not require reading the article in our study, we did borrow guidelines for the roles of driver and navigator as well as encouraged participants to focus on the task at hand (Williams, Wiebe, and Yang, 2002). Moreover, rather than emphasizing efficiency and functionality alone (Nosek, 1998), we instead took the more educational approach (Williams and Kessler, 2001; Williams et al., 2000) knowing that many of our study participants had only been recently exposed to logic approaches as opposed to the few who had more than one previous programming course.

2.4 Programmer Attitude Shifts

Most importantly, our study primarily investigated paired programming's influence on attitudes. Studies suggest significantly increased confidence in program solutions (Mcdowell et al., 2003) as well as satisfaction with the programming process (Domino, Collins, and Hevner, 2007). In addition to measuring changes in confidence and satisfaction levels, we wanted to ascertain whether paired students – particularly, novices – would experience less apprehension in completing programming and other technical tasks.

To measure potential attitude shifts in a challenging technical subject (e.g., programming), we utilized the Attitudes Towards Mathematics Inventory (ATMI) developed by Tapia (1996) and further tested and refined in various studies (Majeed et al., 2013; Sisson, 2011; Tapia and Marsh, 2004). In the instrument, a collection of five-point Likert-scaled questions measure the four subscales of Enjoyment, Motivation, Self Confidence, and Value (Majeed et al., 2013, p. 126) that can be briefly explained as:

- Measure of like or dislike toward the topic (Enjoyment)
- Tendency to engage or avoid the topic (Motivation)
- Belief that one is good or bad at the topic (Self Confidence)
- Belief that the topic is useful or useless (Value)

We also investigated whether participants' attitudes varied according to their assigned roles ("driver" or "navigator") in the study and how this shifted over the experiment. To accomplish this, students were directed to switch roles for the second experiment. As we approached this portion of the study we wanted to address aspects of "cognitive offload," which is the ability for paired teams to solve complex challenges by discussing and working collaboratively rather than attempting solutions alone. Pairs are encouraged to verbalize problems as a means to express thoughts and reasoning processes (Bryant, Romero, and du Boulay, 2008) to determine their solutions whether it be a programming structure or understating of client requirements. In the following section, we discuss our methodology.

3. METHODOLOGY

Our study took place during two weeks of a summer session that permitted extended workshop time in class. During class meetings, 76 students from 4 Information Systems (IS) classes completed a pre-survey to measure their skills and attitudes, then participated in 2 pair-programming exercises in class, and afterward completed a post-survey. Two responses were discarded due to inconsistencies and missing values.

3.1 Student Sample

For our study, we examined how students would approach technical problems tailored to the subject of the particular class (Table 1). Classes were comprised of students from eleven majors, with the majority of non-IS majors in Class A. The demographic profile of participants is shown in Table 2. Class enrollments ranged from first-year to senior students.

Class	Level	Topics	Primary Student Population
Class A: Introduction to Business Computing	First-year course required for all business majors. Can also be used as a university general education course.	Microsoft Office Suite and Basic Web Creation.	First-year business students and various majors outside of the business college using it as a general education course.
Class B: Business Application Programming	Second-year course required of all IS majors.	Java Programming	IS majors.
Class C: Business Analytics I	Second-year course required for all business analytics (BA) majors and minors. Elective for all IS majors and minors.	Advanced Microsoft Access and Excel.	BA majors/minors. IS majors/minors.
Class D: Business Data Mining	Senior-level course required for all BA and IS majors/minors.	Data mining and Analytics (descriptive, predictive, and prescriptive).	BA and IS majors/minors.

Table 1. Courses in the Study

	Frequency	Percentage
Gender		
Female	17	23
Male	57	77
Age		
18 – 24	57	77
25 – 34	16	21.60
>34	1	1.4
Computer Classes Taken (including high school)		
0	2	2.70
1	13	17.60
2 – 3	24	32.40
4 – 6	22	29.70
7 – 10	7	9.50
More than 10	6	8.10

Table 2. Demographic Profile

3.2 Exercises

We tailored each exercise for the class as appropriate. In Class A and Class C, we used identical exercises and software (Excel) which required using Visual Basic for Applications (VBA) to solve simple programming challenges. In Class B, we introduced JFrame challenges to students who previously had coded only for the command line interface (CLI) using the Netbeans Integrated Development Environment (IDE). In Class D, students were asked to use an analytics program (Knime) unfamiliar to them to perform linear regression and classification tree analyses with cross validation.

Before forming pair programming dyads, we measured students' familiarity with the topics by asking questions pertaining to each student's skill level and familiarity. For example, in Class B we asked students how many computing courses previously taken and how many computer languages previously studied. These two questions allowed us to take into account not only established course work in a college setting but also independent, self-motivated study. In Class A and Class C, the pre-survey focused on advanced Microsoft Excel techniques and asked students especially about their familiarity with Microsoft Excel macros. Only one class (Class A) had been exposed to macros via a single Macro Recorder lab. No students had yet written a full VBA program. Finally, in Class D, we recorded students' familiarity with Knime and business analytics. Although Knime was installed on lab computers, it was not utilized in any course students had completed prior to Class D.

Having obtained a baseline of student experience, we created pairs using novice-expert pairing and also randomly assigned about 30% of each class to the control group (individuals) without accounting for experience. Many researchers have noted the benefits of pairing an expert with a novice to increase learning for both; novices learn from experts, and experts enhance their comprehension by tutoring novices (Domino, Collins, and Hevner, 2007; Wray, 2010). In each class, exercises were completed using the same tools or the same technique that had been studied in class sessions immediately prior to the experiment. This was to reduce the likelihood of previous tool use or applied techniques affecting the results (Lui, Barnes, and Chan, 2010).

Students took a pre-survey about one week before participating in the experiments. All experiments were conducted during class times within a single 48-hour period. Students were paired at one system per each team (or individual) at the start of class and then introduced to the concepts of pair-programming using background primarily from the Williams and Kessler (2000) article. About 15 minutes of the session was spent explaining how roles should function.

For Exercise1, the novice student was assigned the navigator role and the experienced student the driver role. Exercise1 was less challenging than Exercise2, and we expected it would be good pair programming practice. After Exercise1 was completed, students submitted their results and recorded start and end times (30-45 minutes total, depending on the class) into our course management system. We then discussed the code, the purpose, etc. in anticipation of Exercise2, and especially to provide basic approaches and concepts to any programming pairs or individuals who did not complete Exercise1.

Exercise2 reversed roles but followed the same procedures. We intentionally designed Exercise2 to be more challenging to measure confidence levels as well as pair jelling (Williams et al., 2000). "Pair jelling" is the pair programming dyad's ability to effectively share the cognitive load of solving complex problems via programming (Bryant, Romero, and du Boulay, 2008). After Exercise2 was completed and recorded, we again discussed potential solutions and then immediately asked students to complete the post-survey in class.

We independently reviewed all exercise submissions and scored each on a scale of 1-5 (with 5 being the highest). Scores of 1 represented an attempt with 3 being an average score. We also noted whether the exercise had been completed (yes/no) and the time it required. If an exercise was not finished, the highest score it could receive was a 4.

3.3 Construct Operationalization

Most of the constructs we used were adapted from existing instruments. The ATMI instrument (Tapia, 1996; Tapia and Marsh, 2004) – designed and implemented to study attitudes toward logic and math subjects – was modified to accommodate the four subject matters (computing, programming, analytics, and data mining) studied in this research. Each question was measured in a five-point Likert-like scale with 1 representing Strongly Disagree and 5 Strongly Agree. Because course topics require some math application and all classes are science, technology, engineering and math (STEM) program components, using ATMI for our purposes seemed reasonable.

Frequently, pair programming is touted to generate certain benefits. Through literature searches, we identified seven outcome (or benefit) variables covering: 1) Skills Enhancement, 2) Enhancement of Self-Esteem, 3) Improved Learning, 4) Feeling Good, 5) Confidence in Subject Matter, 6) Improved Quality of Solution, and 7) Improved Effectiveness in Reaching the Final Solution (e.g., Cockburn and Williams, 2001; Dybå et al., 2007; Nosek, 1998; and others). A question was developed for each of the above seven outcomes. Each question begins, "I believe pairing me with another classmate ____," in which the blank is presented as checkboxes for each of the above seven outcomes, which allows for more than one response per question. As noted in the prior section, completion time for each exercise was recorded, and each solution was evaluated and scored (1-5) by the professor teaching the class.

In order to situate our study within existing research, we utilized approaches from many previous studies, such as providing pair programming background information and approaches, assigning student pairs/dyads based on experience level, monitoring progressively challenging tasks in a classroom environment, and following with post-experiment questions to measure students' reactions to pair programming. However, we added two new items to the research continuum: 1) a diverse set of Information Systems courses ranging from a first-year, non-major course to a senior-level, required IS course, and 2) an attitude measurement tested via a proven instrument primarily used to measure attitude toward mathematics (ATMI) (Majeed et al., 2013; Sisson, 2011; Tapia and Marsh, 2004) that we revised to focus on the programmatic or technical task for each course. Although we found support for increased productivity, functionality, and satisfaction in novice and expert students, we found less change than

anticipated in the sophomore-level, traditional programming course. Support for our findings follows.

4. ANALYSIS AND RESULTS

Before performing exploratory factor analysis (EFA) on ATMI, we first checked several assumptions. Kaiser-Meyer-Olkin (KMO) calculated for the ATMI variables was 0.85, indicating that the sample is adequate for EFA. Bartlett's Test of Sphericity (Chi Squared = 932.512, d.f. = 210, $p < 0.001$) also shows that the correlation matrix of these variables is not an identity matrix. Therefore, the data are appropriate for an EFA. The subjects-to-variables ratio is calculated as 3.52 (74 subjects/21 variables), which is consistent with the literature that suggests a ratio of 2-to-1 (Kline, 1979, p. 166) or 3-to-1 (Arrindell and van der Ende, 1985, p. 166). We then entered these variables into an EFA with varimax rotation (see Table 3). The Chi Square test (Chi Square = 1136.9, d.f. = 132, $p = 0.366$) shows that four factors are sufficient for the model. Items that were heavily cross-loaded in multiple factors were then removed.

The results of EFA presented four dimensions similar to the original ATMI, but questions regarding the confidence dimension clustered into two sub-dimensions. The first confidence dimension, 'Confidence – Visceral reaction,' is for questions related to inward feelings about the subject covered in class. Questions in this dimension were reverse-coded. The second confidence dimension includes questions about the ability to learn, take on intellectual challenges, and other activities. Therefore, it is named 'Confidence – Perceived Ability.' The third dimension, named 'Value,' covers the assessment on the value of the subject matter, and the last dimension, 'Motivation,' encompasses questions regarding respondents' motivations to learn about the subject. Table 3 shows the modified ATMI questions and their factor loadings. The four factors together explain 59.7% of the variance. Construct reliability measured in Cronbach's Alpha for the four factors are 0.939, 0.864, 0.705, and 0.739, respectively.

Modified ATMI Questions	Component			
	Confidence – Visceral reaction	Confidence – Perceived Ability	Value	Motivation
XXX is one of my most dreaded subjects	0.628	-0.307		-0.303
When I hear the words XXX, I have a feeling of dislike.	0.771			
My mind goes blank, and I am unable to think clearly when working with XXX.	0.753			
Studying XXX makes me feel nervous.	0.839			
XXX makes me feel uncomfortable.	0.800			
I am always under a terrible strain in a XXX class.	0.787			
It makes me nervous to even think about having to do a XXX problem.	0.805			
I am always confused in my XXX class.	0.782			
I feel a sense of insecurity when attempting XXX.	0.748			
I am happier in a XXX class than in any other class.		0.534		
I have a lot of self-confidence when it comes to XXX.		0.735		
I am able to solve XXX problems without too much difficulty.	-0.314	0.552		
I expect to do fairly well in any XXX class I take.		0.809		
I learn XXX easily.		0.797		
I believe I am good at solving XXX problems.		0.702		
I can think of many ways that I use XXX outside of school.			0.747	
I plan to take as much XXX as I can during my education.			0.674	
I am willing to take more than the required amount of XXX.		0.376	0.530	
I want to develop my XXX skills.			0.308	0.574
XXX is important in everyday life.				0.672
I get a great deal of satisfaction out of solving a XXX problem.				0.637
Note: XXX refers to the major subject covered in each class. These subjects are programming, introduction to computing, business analytics, and data mining.				

Table 3. Exploratory Factor Analysis on ATMI

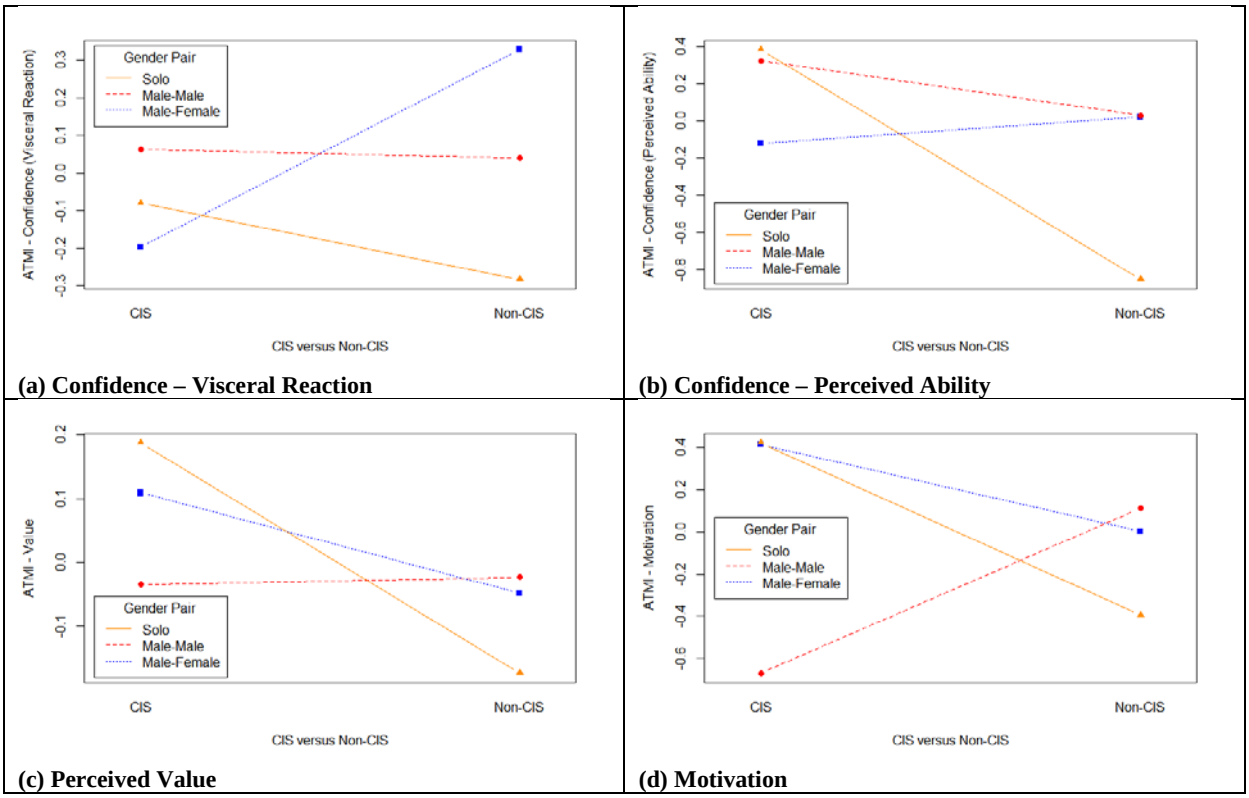
4.1 Demographic Difference on Attitude

We examined the effect of demographic difference on the attitude toward pair programming through a multi-way MANOVA (Table 4). The dependent variables are the four ATMI dimensions. The independent variables include gender pairs, number of computer classes taken, and CIS vs. non-CIS majors. Gender pairs are a categorical variable that includes gender dyads such as male-male, male-female (or female-male), and solo on each team. 'Number of computer classes taken' is discretized into two categories ('many' versus 'few') due to smaller cell sizes for certain categories (e.g., 'None' = 2 and 'More than 10' = 6, as shown in Table 2). The cutoff was 2-3 classes or fewer for the 'few' category. The main effects of the independent variables were not statistically significant, but there was an interaction effect between gender pairs and CIS vs. Non-CIS ($p < 0.05$). Another interaction effect occurred involving all three independent variables ($p < 0.01$).

Because there was an interaction between gender pairs and CIS vs. Non-CIS, Figure 1 illustrates the interaction plots on all four ATMI dimensions individually. Figure 1a shows that non-CIS majors in mixed gender (male-female and female-male) teams had a higher confidence on visceral reaction than CIS majors, thereby triggering the interaction effect. Non-CIS majors working as solo or in male-male teams had the same or slightly lower confidence in visceral reaction than CIS majors. When looking at the perceived level of ability (Figure 1b), non-CIS majors working alone were the only group of non-CIS that had substantially less confidence in perceived ability compared to their CIS counterparts. CIS majors working alone or in male-male pair had a high level of perceived ability compared to mixed gender teams. Non-CIS majors had a lower perceived value when working alone, but CIS majors seemed to prefer working alone (Figure 1c). Figure 1d shows a surprising pattern, in which CIS majors working in male-male teams had significantly less motivation than CIS majors in the other two team configurations. Generally, non-CIS majors had less motivation to learn the subject matter than CIS majors, with the exception of CIS majors working in male-male teams.

Independent Variable	Pillai	df	F	Sig.
Gender pairs	0.080	2	0.628	0.753
Number of computer classes taken	0.112	1	1.856	0.130
CIS vs. Non-CIS	0.084	1	1.355	0.261
Gender Pairs * Computer Classes Taken	0.154	2	1.255	0.273
Gender Pairs * CIS vs. Non-CIS	0.278	2	2.425	0.018 *
Computer Classes Taken * CIS vs. Non-CIS	0.114	1	1.907	0.121
Gender Pairs * Computer Classes Taken * CIS vs. Non-CIS	0.328	2	2.940	0.005 **
* $p < 0.05$, ** $p < 0.01$				

Table 4. Multi-way MANOVA to Assess the Effects of Demographics on ATMI Attitudes



(a) Confidence – Visceral Reaction

(b) Confidence – Perceived Ability

(c) Perceived Value

(d) Motivation

Figure 1. Interaction Plots – Effects of Demographics on ATMI Attitudes

4.2 Predictors of Solution Accuracy/Quality

A hierarchical linear regression analysis was performed to predict the effect of XP. The dependent variable was the score of the second exercise earned by each team or solo participant. Navigators and drivers swapped their roles in this second exercise, allowing navigators to apply what they had just learned from Exercise1 and from the solution demonstration after the first exercise. Results of the second exercise reflect their performance as drivers.

In the first block of hierarchical regression (Model 1 in Table 5), demographic variables (i.e., gender pair and age) were first entered into the equation. Gender pair is a categorical variable dummy-coded into male-female, male-male, and solo variables to study the dynamics of gender interaction in XP. Although the male-male pair had a statistical significance ($p < 0.05$), the predictors did not explain much of the variation of the dependent variable (adjusted $R^2 = 0.05$, $p > 0.05$). This indicates that gender and age are not sufficient to explain the variation in the exercise score.

In the second block (Model 2 in Table 5), four categories of variables were included: 1) the four ATMI attitude dimensions, 2) the dummy-coded class variables, 3) the factor analyzed outcome variables for XP, and 4) other variables (such as computer classes taken, completion of Exercise2, and score difference between the two exercises). The seven outcomes of XP were identified from the literature including 1. Skills Enhancement, 2. Enhancement of Self-Esteem, 3. Improved Learning, 4. Feeling Good, 5. Confidence in Subject Matter, 6. Improved Quality of Solution, and 7. Improved Effectiveness in Reaching the Final Solution. Students were asked to assess their perception on each of these outcomes. Because meanings of these seven outcomes may overlap (e.g., feeling good and self-esteem), the resulting perceived outcomes were factor analyzed with the varimax rotation, which resulted in two factors. The overall factor model explains 61.375% of the variance. The first factor concerns the enhancement of one's perceived expertise (i.e., enhanced perceived expertise), which includes three out of the seven perceived outcomes previously identified: enhances my skills, improves my learning, and improves the effectiveness in reaching the final solution. The second factor (i.e., enhanced sense of accomplishment) measures the enhancement of one's sense of accomplishment resulting from pair programming, which includes three other perceived outcomes: enhances my self-esteem, makes me feel good, and improves the quality of the solution. The last perceived outcome – improves my learning – heavily cross-loaded in both factors and was dropped from the factor analysis.

As the final model in Table 5 illustrates, the effect of the demographic variables from Model 1 changes as other relevant predictors were added. Gender pair did not have a statistical significance on the exercise score, but age did ($p < 0.05$). Of the four ATMI dimensions, only 'Confidence in Perceived Ability' had statistical significance on the exercise score ($p < 0.05$). Among the factor analyzed perceived outcomes, 'Enhanced Sense of Accomplishment' had a negative but statistical significance on the exercise score. Both completion of the exercise and score difference had a positive effect on the dependent variable. However, there was no statistically significant effect of classes on the exercise score. The overall adjusted R^2 value was 0.680, a large improvement ($\Delta R^2 = 0.630$) over the first model. Co-linearity analysis showed no serious concern of multi-collinearity with tolerance values all greater than 0.20.

4.3 Parameters' Influence

In order to determine whether there was a difference across classes on the perceived outcomes of XP, the group means were compared. Although the Levene's Test for Homogeneity of Variance ($p > 0.05$) showed that there was no significant issue with this assumption, the normality assumption through Shapiro-Wilk's Normality Test ($W = 0.96$, $p < 0.05$) was violated. Therefore, non-parametric Kruskal-Wallis' tests are appropriate. As Table 6 shows, there was a mean difference on one's sense of accomplishment among the four classes involved. Post-hoc, pairwise comparison using Dunn's Test is reported in Table 7. Here there is a statistical difference in mean values between the freshman introduction to computing class (Class A) and the senior data mining class (Class D), with the senior class rated higher on sense of accomplishment. There was no statistical significance between Class D and the other two classes (B and C).

Model		Coefficients		t	Sig.	
		B	Std. Error			
1	(Intercept)	2.560	0.845	3.031	0.003	**
	Gender Pair (Male-Male)	0.904	0.364	2.482	0.016	*
	Gender Pair (Male-Female)	0.383	0.394	0.974	0.333	
	Age	0.114	0.323	0.353	0.725	
	F = 2.249, df = 3, R = 0.298, Adjusted R^2 = 0.05, p = 0.09					
2	(Intercept)	3.577	0.659	5.425	0.000	***
	Gender Pair (Male-Male)	0.331	0.244	1.357	0.180	
	Gender Pair (Male-Female)	0.241	0.258	0.935	0.354	
	Age	-0.587	0.221	-2.652	0.010	*
	ATMI – Confidence (Visceral Reaction)	-0.034	0.091	-0.376	0.708	
	ATMI – Confidence (Perceived Ability)	0.214	0.095	2.251	0.028	*
	ATMI - Value	-0.168	0.103	-1.632	0.108	
	ATMI - Motivation	0.126	0.117	1.077	0.286	
	Enhanced Perceived Expertise	0.019	0.100	0.187	0.853	
	Enhanced Sense of Accomplishment	-0.282	0.102	-2.769	0.008	**
	Difference of completion time	-0.029	0.017	-1.685	0.098	
	Computer classes taken	0.082	0.072	1.145	0.257	
	Completion of Exercise2	1.342	0.282	4.758	0.000	***
	Score difference between two exercises	0.437	0.072	6.082	0.000	***
	Class A	-0.556	0.314	-1.774	0.082	
	Class B	0.309	0.296	1.045	0.301	
	Class C	-0.040	0.311	-0.127	0.899	
	F statistic = 13.729, df = 16, R = 0.866, Adjusted R^2 = 0.680 (ΔR^2 = 0.630), p < 0.001					
* p < 0.05, ** p < 0.01, *** p < 0.001						

Table 5. Hierarchical Linear Regression

	Kruskal-Wallis χ^2	df	Sig.
Pair programming enhances perceived expertise	1.90	3	0.60
Pair programming enhances one's sense of accomplishment	8.30	3	0.04**

Table 6. Kruskal-Wallis' Test on the Difference of Perceived Outcome across the Four Classes

(1) Enhanced Perceived Expertise				
	Class A	**Class B**	**Class C**	**Class D**
Class A	-			
Class B	-0.782[+] (0.434/0.651)[++]	-		
Class C	-0.903 (0.367/0.733)	-0.066 (0.947/0.947)	-	
Class D	0.191 (0.848/1.00)	0.994 (0.321/0.961)	1.138 (0.255/1.00)	-
(2) Enhanced Sense of Accomplishment				
	Class A	**Class B**	**Class C**	**Class D**
Class A	-			
Class B	-1.448 (0.147/0.442)	-		
Class C	-1.373 (0.168/0.339)	0.149 (0.882/0.882)	-	
Class D	-2.870 (0.004/0.024)	-1.029 (0.303/0.363)	-1.273 (0.203/0.305)	-
+ Z statistic. ++ p values in this format: (unadjusted p-value / adjusted p-value). P-values were adjusted with the Benjamini-Hochberg method.				

Table 7. Post-hoc Pairwise Comparison – Dunn's Test

5. DISCUSSION

Industry has implemented, and academia has tested, a substantial number of Agile software development approaches and techniques. From this corpus, we sought empirical evidence for extending such a methodology beyond software development. Although there is anecdotal support for the use of Agile methodology in non-IT disciplines, empirical evidence is lacking, particularly as related to the potential effect on participants' attitudes and project outcomes. Our study therefore focuses on how a subset of Agile software development called pair programming can be applied to projects other than software development. In four university IT classes covering introduction to computing, programming, analytics, and data mining, we administered two Agile exercises, with results as discussed below that should benefit future Agile approaches and studies. We found interesting distinctions among CIS and non-CIS mixed gender dyads in terms of confidence levels and visceral reactions to the assigned tasks with non-CIS students higher overall. However, demographics and team mixtures alone do not specifically account for perceived higher levels of performance and outcome. Instead, we found that a sense of accomplishment and completion of the work were more of an indication of increased exercise performance.

5.1 Attitude Impact

The ATMI instrument was adopted to assess attitudes toward pair programming and was administered to the participants two times, one approximately one week before the experiment (t_1) and one immediately after (t_2) to track any changes. The difference of ATMI between t_1 and t_2 was factor analyzed resulting in the following four key dimensions: 1) Confidence about Visceral Reaction, 2) Confidence about One's Own Ability, 3) Value of the Agile Approach, and 4) Motivation to Deepen Adoption or Learning. As the multi-way MANOVA in Table 4 shows, the interaction effect between gender dyads (male-male, mixed gender, and solo) and participants' majors was statistically significant.

Further analysis through the interaction plots reveals several interesting patterns. First, the CIS mixed gender teams appear to have a lower level of confidence on visceral reaction than non-CIS counterparts with the same team composition. The mixed gender teams of non-CIS majors had the highest level of visceral reaction among all non-CIS teams; by contrast, this same team composition among CIS majors had the lowest level of visceral reaction. Second, mixed gender teams had a similar level of perceived ability regardless of their majors (CIS or non-CIS). Third, non-CIS participants working alone had the lowest level of perceived ability. Consequently, this group also had the lowest level of perceived values of pair programming. Fourth, the male-male group of non-CIS majors interestingly had a higher level of motivation to learn than their CIS counterparts.

Existing studies on gender dynamics in the team setting are mixed, with studies showing male-male teams having greater self-efficacy on technical subjects (Hartzel, 2003), no significant difference in performance for mixed gender teams (Kaufman and Felder, 2000), and better performance observed for mixed gender teams than teams dominated by either gender (Hoogendoorn, Oosterbeek, and van Praag, 2013). Our work provides a possible insight into these mixed findings of gender dyads by looking at attitudes (measured by ATMI), which is a construct confirmed in existing information systems theories (e.g., research on technology acceptance model and theory of planned behavior) as the antecedent of behavior. Our four types of ATMI-based attitudes are reasons for the dynamics of behavioral results or performance. As all four sub-figures in Figure 1 show consistent lower levels of attitude for non-CIS

majors working alone, pair programming was helpful for non-CIS majors. Compared to CIS majors, non-CIS majors had a higher confidence on visceral reaction within mixed gender teams and a higher motivation to learn in male-male teams. As for CIS majors, findings are somewhat mixed. Working alone seems to generate a higher perceived level of ability, value, and motivation, yet the same level is observed for mixed gender groups on motivation to learn and for male-male groups on perceived ability. As a result, mixing genders within a team may be associated with a higher level of intention to learn for CIS majors, but a higher level of confidence in visceral reaction for non-CIS students.

5.2 Predictors of Performance

The hierarchical linear regression results also were interesting. Demographics as the only independent variables did not reliably predict the variance of the dependent variable. The resulting equation in Model 1 (Table 5) did not fit the data well (adjusted $R^2 = 0.05$), and only the male-male gender pair had an effect on the dependent variable. As a result, these demographic aspects alone did not seem to explain the variance. After adding variables, including ATMI attitudes, perceived outcomes of pair programming, classes, completion time, and pre-existing knowledge of the subject matter, the R^2 value in Model 2 (Table 5) jumped to 0.680, representing an increase of 0.630. Although pair programming has been associated with higher levels of performance and outcome (e.g., Domino, Collins, and Hevner, 2007), our work shows that this heightened performance is because of higher perceived ability, a sense of accomplishment, and completion of work. Gender compositions of teams did not show a statistical difference on performance.

Interestingly, no class dummy variables influenced performance. Traditionally, pair programming has been applied to software development or related IT projects. Because classes in our experiment covered four different subjects, with only one being programming, this finding indicates that subject matter does not greatly affect performance, and thus provides a rationale for using pair programming in non-hardcore, IT classes. Coupled with our previous finding on perceived ability, sense of accomplishment, and completion of work, our work parallels existing studies (Merisalo-Rantanen, Tuunanen, and Rossi, 2005) in that experience and expertise are keys to success in XP. Therefore, class subjects alone do not determine performance in pair programming; rather, the ability in mastering the subject is what matters. It is important to point out that ability to perform had a statistical significance in the present study, but perceived expertise did not. As a result, practitioners are recommended to first target ability cultivation when applying pair programming. Even if participants have not attained mastery in skills, the perceived level of enhanced ability matters more.

5.3 Confidence Impact

Of the two perceived outcome factors, the second (pair programming enhances one's sense of accomplishment) had a high, but negative, influence on performance. This is unexpected; typically, the correlation would be positive. However, variables that measure accomplishment more broadly include enhanced self-esteem and satisfaction. A possible interpretation is that students derived accomplishment not only from results but also from intangibles, such as the opportunity to learn new concepts and work with new tools. Future researchers may want to more closely examine sources of accomplishment (e.g., accomplishment from immersion in the process versus accomplishment from achieving the best result).

5.4 Study Contributions

This study contributes to the literature in several ways. First, influences of one's attitude has generally received little attention in Agile literature. Our work indicates that pair programming is not necessarily a standalone catalyst for attitude change. Our data shows a direct statistical effect of one ATMI attitude dimension (perceived ability) on performance. This provides a new direction for research into the role of pair programming, especially for those who see little or no attitude shift with this methodology. Although we uncovered some beginning understanding of attitudes on performance, future researchers may want to study triggers of attitude change in the Agile methodology.

Second, we found that pair programming does not improve all types of outcomes. In the present study, seven outcome variables derived from prior research (e.g., Cockburn and Williams, 2001; Dybå et al., 2007; Nosek, 1998) were factor analyzed showing two distinctive factors (enhancement of perceived expertise and enhancement of one's sense of accomplishment). Our findings indicate that only enhancement of one's sense of accomplishment was a key predictor of performance or solution quality, but enhancement of one's expertise was not. Our approach in examining outcome variables together with attitudes for their effect on solution quality is helpful because prior research either primarily focused on limited number of outcomes or paid little attention on how perceived outcomes together with attitudes affect performance.

Third, subject matter is less significant to solution quality improvement than participation in pair programming. This is demonstrated by the lack of statistical significance in all dummy-coded class variables. As our participants represented different academic majors and different levels of academic preparation, our findings support that the programming course was no different from other non-hardcore, IS courses on exercise performance. As a result, one does not have to be in a software development course to enjoy the benefits of pair programming.

Finally, perceptions of outcomes vary across levels of academic preparation. Our series of Kruskal-Wallis non-parametric tests (Table 6) and the post-hoc analysis (Table 7) show that Class A (an introduction to computing class typically taken by freshmen) was distinctively lower than Class D (primarily comprised of seniors) in their sense of accomplishment through pair programming. One explanation is that less academic preparation could limit participants' abilities or options for techniques to craft a solution. Therefore, experience in formulating a solution also matters. Compared to prior studies that focus primarily on domain specific experience, this finding also contributes to the body of knowledge in that experience in problem and solution formulation could help, especially when domain specific knowledge is lacking. Future research could offer further insights in this direction.

5.5 Study Limitations

This study offers only a brief appraisal of the Agile effect on attitude and performance. Our focused, in-class exercises were not designed to address the "learning effect," in which participants learn to cooperate over time. However, mixed gender dyads did result in higher levels of intention to learn for CIS majors, but less in non-CIS majors, so this needs to be explored further. Future research might consider what type of training should be implemented to minimize this difference before participants engage in Agile methodology. Many studies attribute learning and problem solving to the novice/expert experience gap, but this conclusion may be too simplistic as mixed dyads in non-CIS students demonstrated higher levels of confidence in visceral reactions to the process and more perceived satisfaction overall.

Second, because our work focuses on IT education using students, we cannot generalize the findings to other settings (such as the work environment). However, using four subject matters and participants from eleven academic majors does offer a greater potential for generalizability than using a single subject or discipline. Finally, because the experiment was conducted in a computer lab in a face-to-face setting, extending our finding to other settings, such as virtual environments, demands caution.

6. CONCLUSION

Pair programming, or its relative, Agile development, originates from IS. Several decades of interest and study have expanded our understanding of what both techniques can contribute to industry or education. Our research adds to this scholarship as it relates to IS education. First, the concept of pair programming may be applied even in IS classes that do not require programming. Our findings show improved levels of confidence and interaction in non-CIS students, indicating that Agile approaches are compatible with active learning pedagogies. Even in traditional IS curricula, not all courses are about or require software development; examples include networking, analytics, IS strategy, and project management. Our work offers beginning empirical evidence endorsing the application of pair programming in non-programming classes.

Second, our work affirms the merits of providing adequate dynamics for task-related knowledge. We recommend the consideration of effective team combinations – to include gender dyad formation – especially for active learning opportunities or work assignments for which the students formerly have not had direct domain knowledge.

Finally, confidence has been identified as key for the success of pair programming (Koriat, 2012); however, our findings suggest that not all types of confidence have an effect on performance. Therefore, we recommend that pair programming research focus on multiple forms of attributes (such as ATMI dimensions reported in the present study) in addition to outcome variables to better document the benefits of pair programming and make a stronger case for overall learning.

Although we fully support and promote pair programming and Agile techniques, we would caution that these are not a panacea to learning. More should be done to explore the various facets that will help all students – IS or otherwise – learn to function in an increasingly technical world.

7. REFERENCES

Arrindell, W. A. & van der Ende. J. (1985). An Empirical Test of the Utility of the Observations-to-Variables Ratio in Factor and Components Analysis. *Applied Psychological Measurement*, 9, 165-178.

Bryant, S., Romero, P., & du Boulay, B. (2008). Pair Programming and the Mysterious Role of the Navigator. *International Journal of Human Computer Studies*, 66(7), 519-529.

Canfora, G., Cimitile, A., Garcia, F., Piattini, M., & Visaggio, C. A. (2007). Evaluating Performances of Pair Designing in Industry. *Journal of Systems and Software*, 80(8), 1317-1327.

Cockburn, A. & Williams, L. (2001). The Costs and Benefits of Pair Programming. *eXtreme Programming and Flexible Processes in Software Engineering XP2000*, 223-243.

Domino, M. A., Collins, R. W., & Hevner, A. R. (2007). Controlled Experimentation on Adaptations of Pair Programming. *Information Technology and Management*, 8(4), 297-312.

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

Flor, N. (1998). Side-by-Side Collaboration: A Case Study. *International Journal of Human-Computer Studies*, 49(3), 201-222.

Hartzel, K. (2003). How Self-Efficacy and Gender Issues Affect Software Adoption and Use. *Communications of the ACM*, 46(9), 167-171.

Hoogendoorn, S., Oosterbeek, H. & Praag, M. V. (2013). The Impact of Gender Diversity on the Performance of Business Teams: Evidence from a Field Experiment. *Management Science*, 59(7), 1514-1528.

Kaufman, D. B. & Felder, R. M. (2000). Accounting for Individual Effort in Cooperative Learning Teams. *Journal of Engineering Education*, 89(2), 133-140.

Kline, P. (1979). *Psychometrics and Psychology*. London: Academic Press.

Koriat, A. (2012). When are Two Heads Better than One and Why? *Science*, 336(6079), 360-362.

Lui, K. M., Barnes, K. A., & Chan, K. C. C. (2010). Pair Programming: Issues and Challenges. *Agile Software Development*, 143-163.

Lui, K. M. & Chan, K. C. C. (2006). Pair Programming Productivity: Novice-Novice vs. Expert-Expert. *International Journal of Human Computer Studies*, 64(9), 915-925.

Majeed, A. A., Gusti, I., Darmawan, N., & Lynch, P. (2013). A Confirmatory Factor Analysis of Attitudes Toward Mathematics Inventory (ATMI). *The Mathematics Educator*, 15(1), 121-135.

Mcdowell, C., Werner, L., Bullock, H. E., & Fernald, J. (2003). The Impact of Pair Programming on Student Performance, Perception and Persistence. *Proceedings of the 25th International Conference on Software Engineering*, 602-607.

McDowell, C., Werner, L., Bullock, H. E., & Fernald, J. (2002). The Effects of Pair-Programming on Performance in an Introductory Programming Course. *ACM SIGCSE Bulletin*, 34(1), 38.

Merisalo-Rantanen, H., Tuunanen, T., & Rossi, M. (2005). Is Extreme Programming Just Old Wine in New Bottles: A Comparison of Two Cases. *Journal of Database Management*, 16(4), 41-61.

Nagappan, N., Williams, L., Ferzli, M., Wiebe, E., Yang, K., Miller, C., & Balik, S. (2003). Improving the CS1 Experience with Pair Programming. *ACM SIGCSE Bulletin*, 35(1), 359.

Nawrocki, J. R., Jasinski, M., Olek, L., & Lange, B. (2005). Pair Programming vs. Side-by-Side Programming. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (Vol. 3792 LNCS, 28-38).

Nosek, J. T. (1998). The Case for Collaborative Programming. *Communications of the ACM*, 41(3), 105-108.

Sanders, D. (2002). Student Perceptions of the Suitability of Extreme and Pair Programming. *Extreme Programming Perspectives*, 168-174.

Sherrell, L. B. & Robertson, J. J. (2006). Pair Programming and Agile Software Development: Experiences in a College Setting. *Journal of Computing in Small Colleges*, 22(2), 145-153.

Sisson, L. H. (2011). Examining the Attitudes and Outcomes of Students Enrolled in a Developmental Mathematics Course at a Central Florida Community College. *ProQuest Dissertations and Theses*, 206-n/a.

Stephens, M. & Rosenberg, D. (2004). The Irony of Extreme Programming. *Dr. Dobb's Journal*, 29(5), 44-47.

Tapia, M. (1996). The Attitudes toward Mathematics Instrument. *Paper presented at the Annual Meeting of the Mid-South Educational Research Association*, 1-19, Tuscaloosa, AL.

Tapia, M. & Marsh, G. E. (2004). An Instrument to Measure Mathematics Attitudes. *Academic Exchange Quarterly*.

Tingling, P. & Saeed, A. (2007). Extreme Programming in Action: A Longitudinal Case Study. *Human-Computer Interaction. Interaction Design and Usability*, 242-251.

Vanhanen, J. & Mäntylä, M. V. (2013). A Systematic Mapping Study of Empirical Studies on the Use of Pair Programming in Industry. *International Journal of Software Engineering and Knowledge Engineering*, 23(9), 1221-1267.

Williams, L. A. & Kessler, R. R. (2000). All I Really Need to Know about Pair Programming I Learned in Kindergarten. *Communications of the ACM*, 43(5), 108-114.

Williams, L. A. & Kessler, R. R. (2001). Experiments with Industry's "Pair-Programming" Model in the Computer Science Classroom. *Computer Science Education*, 11(1), 7-20.

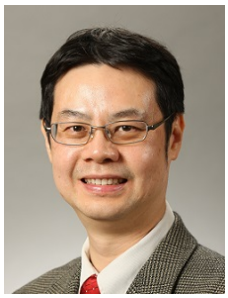
Williams, L., Kessler, R. R., Cunningham, W., & Jeffries, R. (2000). Strengthening the Case for Pair Programming. *IEEE Software*, 17(4), 19-25.

Williams, L. & Upchurch, R. L. (2001). In Support of Student Pair-Programming. *SIGCSE Bulletin*, 33(1), 327-331.

Williams, L., Wiebe, E., & Yang, K. (2002). In Support of Pair Programming in the Introductory Computer Science Course. *Computer Science Education*, 3(12), 197-212.

Wray, S. (2010). How Pair Programming Really Works. *IEEE Software*, 27(1), 50-55.

AUTHOR BIOGRAPHIES



Dr. Kuanchin Chen is a Professor of Computer Information Systems and John W. Snyder Faculty Fellow at Western Michigan University. His research interests include electronic business, Big Data, social networking, project management, privacy & security, online behavioral issues, business analytics, data mining, and human computer interactions. He has published in journals including *Information Systems Journal*, *Decision Support Systems*, *Information & Management*, *Journal of Database Management*, *Internet Research*, *Communications of the Association for Information Systems*, *Electronic Commerce Research and Applications*, *Journal of Global Information Management*, *DATA BASE for Advances in Information Systems*, *IEEE Transactions on Education*, *Decision Sciences Journal of Innovative Education*, *Journal of Computer Information Systems* and many others. He is a member of several journal editorial or advisory boards. Dr. Chen is also a recipient of several research and teaching awards, including awards given by scholarly journals & conferences, department, college, university, and U.S. Fulbright program. Dr. Chen has frequently been invited to give research talks at universities, government agencies, and other institutions.

Dr. Alan Rea is a Professor of Computer Information Systems in the Department of Business Information Systems at the Haworth College of Business, Western Michigan University. Teaching courses in information security and object-oriented programming, Dr. Rea integrates free and open source software and whenever possible, Agile approaches, to accommodate the dynamic environment within information systems. His research concentrates on secure application and system development as well as organizational information assurance and risk management approaches. In particular, he has examined security and privacy implications associated with developing, deploying, and managing web and mobile applications as well as Internet of Things devices. Dr. Rea also co-leads a cross-disciplinary, grant-funded initiative to develop a mobile application assisting in concussion recovery and education, particularly for sports-related injuries among young adults. His research has been published in the *Journal of Information Systems Education*, *International Journal of Electronic Healthcare*, *Journal of Information Systems Security*, *Journal of Information Privacy and Security*, *Journal of Computer Information Systems*, *Communications of the ACM*, and *Journal of Digital Forensics, Security and Law*. Currently he co-directs a cross-disciplinary M.S. and Graduate Certificate in Information Security.

Information Systems & Computing
Academic Professionals

STATEMENT OF PEER REVIEW INTEGRITY

All papers published in the Journal of Information Systems Education have undergone rigorous peer review. This includes an initial editor screening and double-blind refereeing by three or more expert referees.

ISSN 2574-3872