Journal of the Midwest Association for Information Systems (JMWAIS)

Volume 2018 | Issue 1 Article 3

Comparing Test-Driven Development and Pair Programming to Improve the Learning of Programming Languages

Michael A. Eierman
University of Wisconsin Oshkosh, eierman@uwosh.edu

Jakob Iversen
University of Wisconsin Oshkosh, iversen@uwosh.edu

Journal of the Midwest Association for Information Systems

Volume 2018 | Issue 1 Article 3

Date: 01-31-2018

Comparing Test-Driven Development and Pair Programming to Improve the Learning of Programming Languages

Michael A. Eierman

University of Wisconsin Oshkosh, eierman@uwosh.edu

Jakob H. Iversen

University of Wisconsin Oshkosh, iversen@uwosh.edu

Abstract

This article explores student perceptions of the impact that test-driven development (TDD) and pair programming has on their ability to learn programming. In particular, we examine how test-driven development compares to pair programming in student's perceptions. The basis of the study is a survey of students who have completed two programming courses that use the C# programming language and use both pair programming and test-driven development techniques to support learning of object-oriented programming. The results indicate that both pair programming and TDD are considered helpful by students but TDD is seen as the more valuable practice.

Keywords: Programming education, pair programming, test-driven development

DOI: 10.17705/3jmwa.000038

1. Introduction

Agile practices have been widely adopted by industry as an effective approach to software development (Hassan, Iqbal, Sattar, & Rafi, 2015; Kumar & Bhatia, 2012; Monett, 2013). The teaching of agile methodologies is now an expected component of computer science and information systems programs (Monett, 2013). There are various approaches to incorporating agile in the curriculum including stand-alone courses on the approach, project-based courses, and courses focused on learning programming. An increasing focus for research is exploring the potential impact of incorporating agile practices as part of learning programming has on learning outcomes. Past research has found promising evidence that using pair programming significantly improved student performance (Bipp, Lepper, & Schmedding, 2008; Williams, Wiebe, Yang, Ferzli, & Miller, 2002).

Learning to program is difficult for many students (Isong, 2014). It involves learning a new language, learning a new way to break down and solve a problem, and requires an exactness and attention to detail that they may have never had to apply beyond math courses. Programming can cause cognitive overload (Brusilovsky, Kouchnirenko, Miller, & Tomek, 1994) and students have problems reading, tracking, writing and designing simple code fragments (Rosminah, md derus, & Ali, 2012). Many of the modern Integrated Development Environments (IDEs) provide support for learning the language and syntax through code completion. However, there is little support for breaking down and solving the problem. Often, beginning programmers indicate that while they understand everything in lecture, and understand what the assignment requires them to do, when they actually start to write the code they don't know where to start or what they should do next. This leaves them frustrated and unable to make progress until the professor intervenes (if they even alert the professor). Too often, the IDE becomes a crutch and a potential obstruction to learning because it is easy for the student to just throw code against the wall and see what sticks, rather than thinking through the problem in a structured way. This may lead to a working program, but often results in solutions that are brute force and demonstrate a lack of understanding of how to actually code the solution. This has become clear to us as our students transition from their first programming course to the second. Students that successfully complete the first course are suddenly extremely challenged when they begin the second course where the focus shifts from creating simple programs with a visual interface to a focus on object-oriented principles implemented in programs without a graphical user interface.

In our curriculum, the first programming class is taught in a traditional lecture-based approach and the second incorporates agile practices. The second course includes pair programming and test-driven development as fundamental approaches to the development of programming assignments. Both courses use the C# programming language. Many studies have demonstrated the efficacy of using pair programming to improve student learning of programming (Umapathy & Ritzhaupt, 2017). Other agile methods have also been found to improve learning (Hassan et al., 2015; Monett, 2013). However, the agile practice of Test-driven Development (TDD) has received relatively little attention in determining its impact on student learning of programming. Our hypothesis is that using test-driven development and writing tests first will provide the focus and direction that students need to both complete the assignment and learn programming better and that this practice will compare favorably with the impact of pair programming on learning. The purpose of this study is to examine student's perceptions of the impact that test-driven development and pair programming has on their ability to learn to code.

The rest of this article is organized as follows. The next section reviews the relevant literature, followed by a description of how agile practices are incorporated into our courses. We then describe how the study was conducted, followed by the results. The final section discusses the implication of the results and provides future research direction.

2. Literature Review

With Agile practices increasingly being employed in industry (Hassan et al., 2015; Kumar & Bhatia, 2012; Monett, 2013), educators have begun incorporating the teaching of this approach to software development into their development curriculum. From there, the examination of the impact of these practices is a natural extension. Early studies suggest there is an impact. For example, Williams et al. (2002) in an experiment with students in a senior software engineering course showed that students working in pairs produced higher quality code, more quickly than those who worked individually. For beginning programmers there also appears to be a benefit. McDowell et al. (2002) found that students in a beginning programming course were more likely to stay in the course, pass exams, and continue on with computer science courses when they worked in pairs rather than alone.

Other studies have corroborated these early findings. Bipp, Lepper, and Schmedding (2008) found that pair programming may lead to more knowledge about software development. Perera (2009) found that Agile and pair programming have a significant impact on student programming skill improvement and improvement of weaker student programming skills. Simon & Hanks (2008) found that when students used pair programming they thought they got stuck less often and explored more ideas and that it generally helped them in a first course in computer science. Kathuria & Goel (Kathuria & Goel, 2010) determined that when inexperienced programmers first completed an individual task and then worked as a pair on a task that combined the individual tasks the inexperienced students showed a marked improvement in performance as compared to experienced students solo programming. This included enhanced problem-solving skills and improvement in the quality of work. In a study of student perceptions of pair programming, Faja (2014) found that while students perceived that their learning, quality of work, and enjoyment was higher with paired programming than their perception of increased productivity. Taken together, the implication of these studies is that pair programming potentially improves student learning and knowledge of software development.

Others have suggested that Agile can be successfully integrated into project-based classes. Monett (2013) found that incorporating Agile practices and Extreme Programming (XP) in a project-based course improved the learning of Agile. Hassan et al. (2015) found that Scrum can be successfully used as a teaching methodology in a capstone course. Isong (2014) suggests that Agile practices and pair programming principles can improve understanding, engagement, and motivation in first year computer science students. He suggests that lecture-based courses do not engage students enough to achieve their learning outcomes.

Test-driven Development is also an important aspect of Agile development. Whereas a number of studies have explored how TDD can be used to increase students' appreciation of the importance in testing (Erdogmus, Morisio, & Torchiano, 2005; Janzen & Saiedian, 2008; Lappalainen, Itkonen, Isomöttönen, & Kollanus, 2010; Spacco & Pugh, 2006), its impact on student learning has not been explored as much as pair programming. In an action research project targeting an introductory programming class, Keefe, Shard, and Dick (2006) explored four eXtreme Programming practices (pair programming, test-driven development, simple design, and refactoring) and found that TDD was the most difficult of the four for students. On the other hand, a series of experiments at California Polytechnic State University (Hilton & Janzen, 2012; Janzen & Saiedian, 2008) using a dedicated web-based tool that enables instructors to create labs based on test-driven learning showed that students were able to learn programming more easily compared to the traditional approach. However, using test-driven development in courses can be challenging and care must be taken in implementing it in the classroom to be effective (Mugridge, 2003).

While the research suggests that Agile and pair programming can help student learning, the evidence is less strong for TDD. Introducing test-driven development can help students in multiple ways as they are learning to program. With TDD, the focus is first on writing the client code to perform some action, which leads to thinking about higher-level design and what problems are being solved, rather than focusing on the implementation details. This should help students not get bogged down in more complicated code that they may have problems writing, as well as help them to break down problems into smaller, more manageable chunks.

Test-driven Development, as practiced through the various unit testing frameworks such as NUnit and JUnit also lends itself to a focus on learning object oriented principles of instantiating objects and calling methods on those objects as a way to create the system. This helps students to realize the importance of object orientation and how to create object oriented systems. This paper investigates whether or not students believe that pair programming and TDD helped them learn programming in C#.

3. Teaching Agile in the Curriculum

The information systems program at University of Wisconsin Oshkosh currently has about 55 students, most of which are traditional undergraduates who have transitioned directly from high schools within the state of Wisconsin. Gender balance is about 65% male and 35% female.

The core IS curriculum includes two required programming courses. The first course (IS 201) is an introduction to business programming using C#. It is taught in a traditional lecture-based format. The second course (IS 318), the primary focus of this study, continues with C# but focuses on the use of Agile methods in programming. Our IS curriculum has two tracks (Enterprise Resource Management and Web and Mobile Development). The development track includes a third C# class (IS 432) focused on web development, as well as a mobile app development class (IS 433) that introduces students to native mobile development for both Android and iOS.

Table 1 shows the schedule for the second required programming course. As can be seen, agile practices are introduced early with pair programming introduced in week 2 and test-driven development introduced in week 3. Once introduced, these practices

are used throughout most of the course. The course has six major assignments, all of which are completed using pair programming. Partners stay together for two assignments before rotating to new partners (Srikanth, Williams, Wiebe, Miller, & Balik, 2004). Pair partners for the first four assignments are assigned by the instructor based on information collected through the Pair Evaluate server from NC State (Williams, 2010). For those first assignments, partners are assigned based on having common work ethic and as much difference as possible on the Myers-Briggs sensing-intuition personality type measures (Katira et al., 2004). For the last two assignments, students are allowed to choose their own partner. Throughout the semester, students are given weekly exercises and tutorials that they are mostly asked to complete on their own – except for those exercises that are connected to learning pair programming and one on shared code. Students complete peer evaluations at the end of each of the six larger assignments with an impact on the grade of the other person in the pair.

Week	Topic
1	C# Review Objects and Classes Including how to draw UML classes and objects
2	Pair programming More objects: Instantiation, calling methods on objects. Encapsulation. Examining object states
3	**Testing**: Whitebox, blackbox. Equivalency Partitioning + Cause-Effect Graphing Test-driven Development Debugging + Exceptions
4	Practice Makes Perfect
5	**Inheritance** + polymorphism + interfaces **UML class diagrams** (inheritance hierarchies)
6	Collections UML Class diagrams (aggregation and association) **Version Control** and Shared Code TFS, Git, VisualStudio.com
7	Tuesday: Practice Makes Perfect Thursday: **Midterm**
8	**Sorting collections** - use to demonstrate interfaces Generics
9	MVC + GUI
10	Practice Makes Perfect
11	**Persistency**: Text file + serialization + XML
12	Plan-based vs. agile User stories
13	Tues: Practice Makes Perfect Thurs: Final Exam
14	Tues: Practice Makes Perfect Thurs: Programming Exam

Table 1. Weekly schedule for IS 318 Agile Application Development in C#.

Test-driven development is used as the predominant approach to create programs, except for the GUI assignment. The programming exam given on the last day of class is a hands-on test completed during class time where students are given a project with tests already completed, but the code to make the tests pass is missing. The students are then tasked with writing the code that will make the tests pass. They are also asked to include a few tests of their own.

Figure 1 shows the view in Visual Studio after students have run the tests for the first time in the TDD-based programming exam. All the tests are failing – primarily due to methods throwing NotImplementedExceptions. As students implement methods they are able to run the tests and get immediate feedback on whether they have successfully implemented the method.

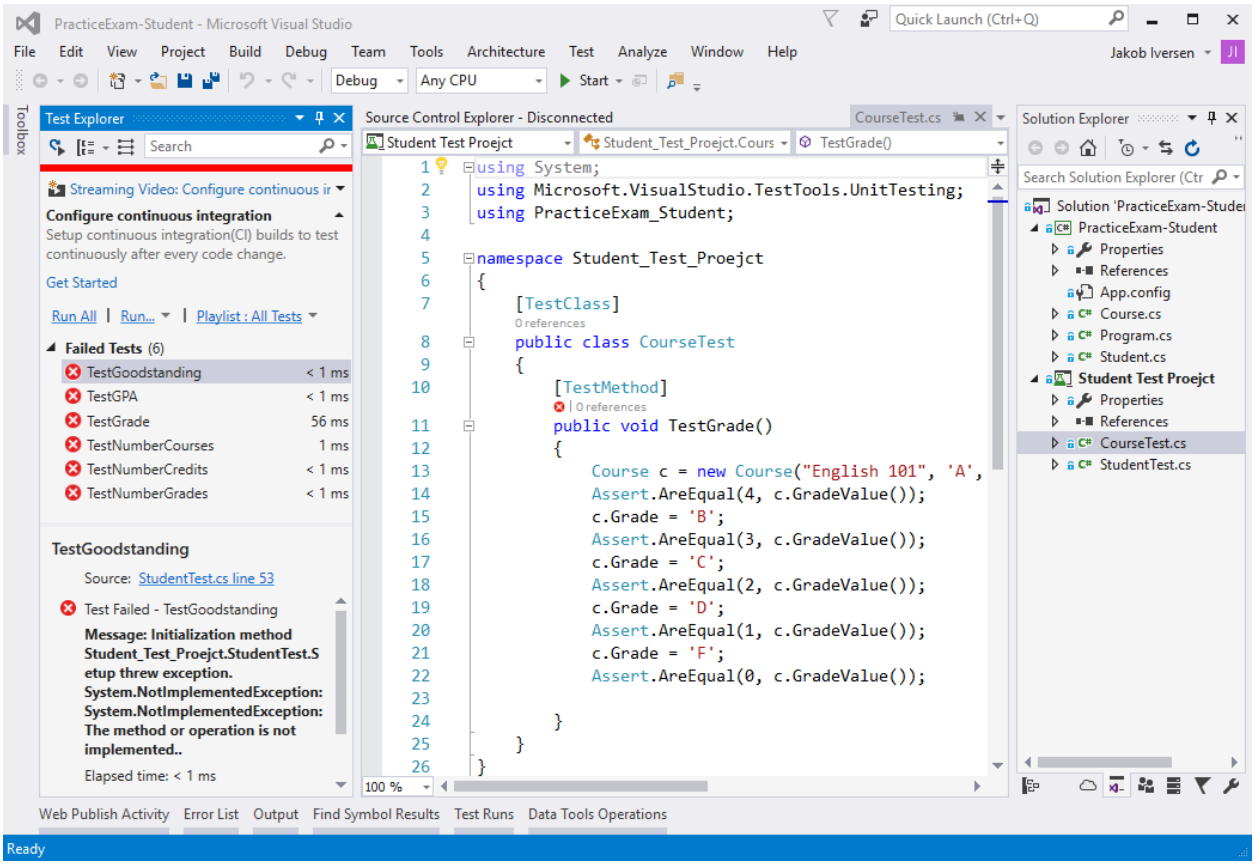

Figure 1. Initial run of tests in TDD-based exam.

Grading of the TDD-based exam is done manually as students are given credit for getting close to the correct answer.

4. Research Method

We administered a survey of the students who had taken the IS 318 course described above during the 2017 spring semester. The students were a mix of junior and senior IS and computer science students, and a mix of genders. To increase response rates, students were given extra credit for completing the survey. A total of 29 surveys were completed. Students were told that only whether they completed the survey or not would be shared with their instructor. The response rate was almost 100%. In the survey, students were asked if pair programming and test-driven development practices had improved their understanding of the programming language, improved their code, and decreased the amount of time spent on assignments. Students were also asked if they would continue using these practices on their own and if they preferred the teaching approach to the traditional lecture based approach used in the first programming class.

All questions were answered on a 5-point Likert scale. The scale was anchored from Strongly Disagree (1) to Neither Agree or Disagree (3) to Strongly Agree (5). The survey questions are included with the results in Table 2.

The same survey was given to students who had taken the course one year earlier during spring 2016. This group of students were subsequently enrolled in the third C# course (IS 432). These students were also given extra credit for completing the survey. This group completed nine (9) surveys and again with almost 100% response rate. The significantly lower number is due to the fact that the Web and Mobile Development emphasis is less popular than the ERP emphasis so fewer students take the course.

5. Results

The results of the survey for students taking the second programming class (IS 318) and the third programming class (IS 432) are presented in Table 2. The mean response was tested to determine if it was significantly different from the neutral response (3) and if the mean score was different for each question between test-driven development and pair programming. Test-driven development was generally perceived to support the student's learning of the programming language. However, this was not as strong for the students who went on to take the third class where TDD was not enforced. While students in the second class were neutral on enforcing the use of TTD on all programming assignments students in the third class were mildly against enforcing it. Students in both classes felt that TDD improved their code and reduced the number of bugs in it. However, neither group agreed that TDD increased the speed of their programming effort. Students in the second class wanted to learn more about the practice and believed it should be used in other programming courses. However, students in the third class did not want either of these. Students in both classes generally believed that TDD was more helpful than pair programming.

Pair programming was also somewhat perceived to support the student's programming effort by students in the second course. However, this result was not as strong as it was for TDD. Students did not think it helped them learn the programming language but did help them reduce the number of bugs and improve the quality of the code. In contrast to TDD, they were not interested in learning more about pair programming. However, they do believe it should be used in other courses. These results are completely different for students in the third course. They had a strong negative view of all aspects of pair programming.

In general, students in both classes thought that learning agile practices, specifically TDD and pair programming would help them in their professional careers and they learned a lot in the class. However, they did not think that the approach to teaching and learning in the course was significantly better than the traditional lecture based approach.

Figure 2 shows the results of the survey questions that are the same for both pair programming and test-driven development. The figure includes one bar for each set of questions for the two student groups, so TDD-318 is the results of the TDD questions for just the students who only took IS 318. This shows clearly that the students who only took IS 318 were consistently more positive, and that Test-driven Development was regarded more positively than Pair Programming.

Exceptions to those general results include that IS 318 students found that pair programming did more to decrease time and improve quality. Both student groups are more likely to continue with pair programming than with test-driven development.

Figure 3 shows the results of the more general questions compared across the two student populations. Both groups found it helpful to their future career to have learned agile practices, and found that they learned a lot in the course. However, the 432 group slightly preferred the non-agile approach employed in the first programming class in the sequence.

Because students in the second course had a more negative view of TDD we examined scores on the TDD exam for both populations in order to understand if this view was due to a learning problem associated with TDD. Figure 4 shows the results of each of the questions on the TDD exam between the two semesters. The students who took both IS 318 and IS 432, took this exam in spring 2016, and those who took only IS 318, took it in spring 2017. The students who took the exam in spring 2016, performed significantly better on several of the questions. Overall, the spring 2016 class had an average score of 54.4 and the spring 2017 class had an average of 44.67. This suggests that students who had a more negative view of TDD had at least as much ability with it as students who had a more favorable view.

Test-driven Development	**Mean (IS 318) N=29**	**Mean (IS 432) N=9**
…helped me better understand programming in C#.	4.0345♦*	3.7778
…decreased the amount of time spent programming my assignments.	2.7931	2.4444*
…reduced the number of bugs I discovered during the development of my programming assignments.	3.6429*	3.7778♦*
…improved the quality of my programming assignments.	3.8621*	3.6667
Developing the test cases prior to programming the system is a practice I will continue on my own.	3.4138	2.8889
I would like to learn more about Test-driven Development.	3.4483♦*	3.1111
I don't think learning about Test-driven Development helped me in any way.	2.0357*	2.4444
I believe Test-driven Development should be used in other programming courses as well.	3.8621*	3.4444
Test-driven Development should be enforced on all programming assignments.	3.0357	2.3333*
Pair Programming		
…helped me better understand programming in C#.	3.4483♦	2.5556
…decreased the amount of time spent programming my assignments.	3.1724	2.1111
…reduced the number of bugs I discovered during the development of my programming assignments.	3.6207*	2.5556♦
…improved the quality of the code I developed.	3.9655*	3.1111
…helped me get a better grade in this course.	3.5862*	2.6667
…made it easier to learn the programming language.	3.3103	2.3750
Pair Programming is a practice I will continue on my own if possible.	3.3448	2.2222
I would like to learn more about Pair Programming.	3.0345♦	2.5556
I don't think learning about Pair Programming helped me in any way.	2.2500*	3.2222
I believe Pair Programming should be used in other programming courses as well.	3.5862*	2.6667
Pair Programming should be enforced on all programming assignments.	2.6552	1.5556
Other Questions		
Learning agile practices will be helpful in my future IS career	4.0000*	4.1111*
Having learned pair programming and test-driven development, I feel better prepared to work as a professional developer.	3.6897*	3.3333
I learned a lot in this course.	4.2414*	4.0000
The approach to teaching and learning in this course helped me learn the material better than the approach to teaching and learning used in the first programming course (INFO SYS 201).	3.4815	2.8571
I did not like the approach to teaching and learning used in this course.	2.7407	3.0000

Table 2. Results of the survey.

♦*-significantly different from the corresponding question on the other practice (Pair Programming vs. Test-driven Development)*

**- significantly different from neutral response (3)*

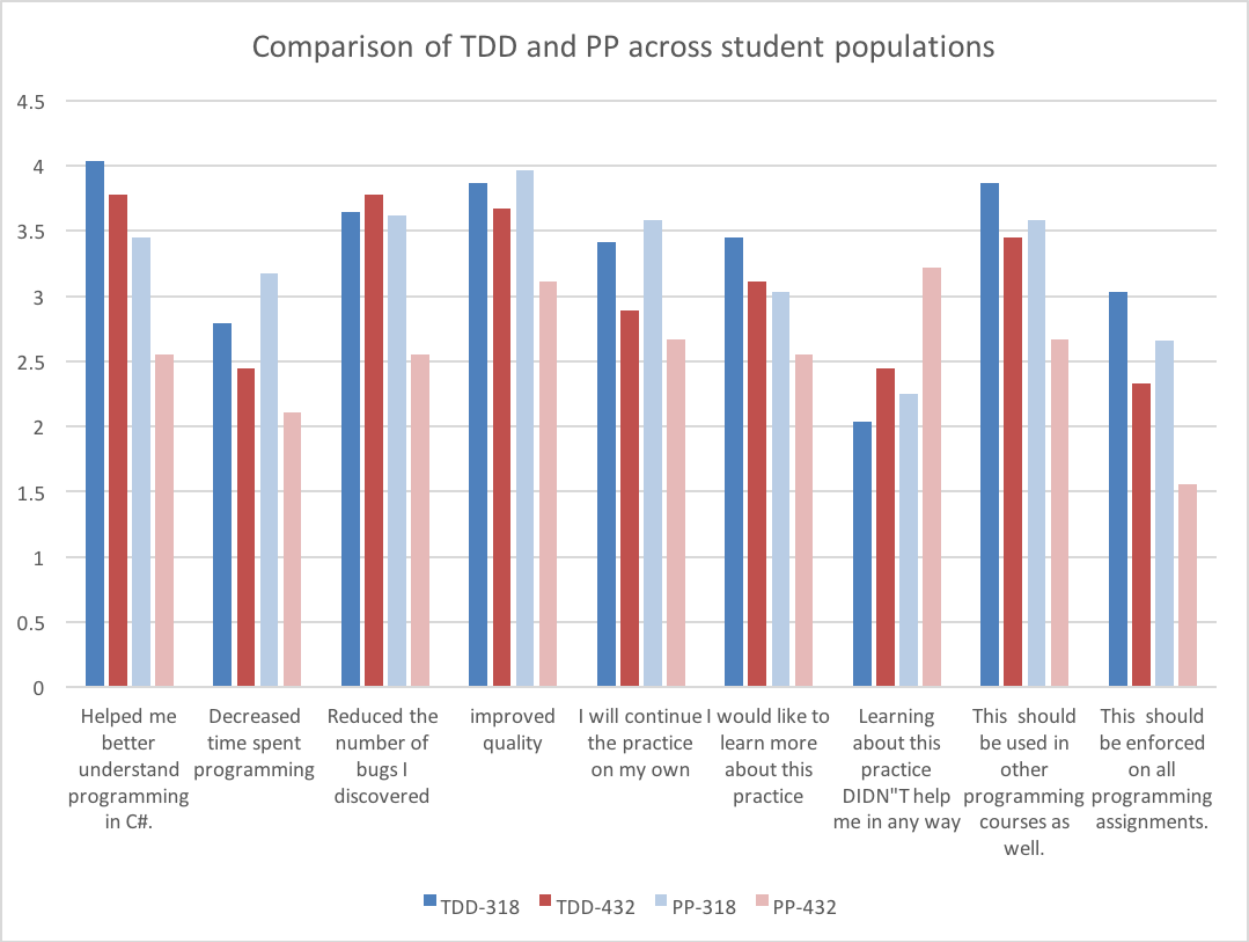

Figure 2. Comparison of Test-driven Development and Pair Programming across both student populations.

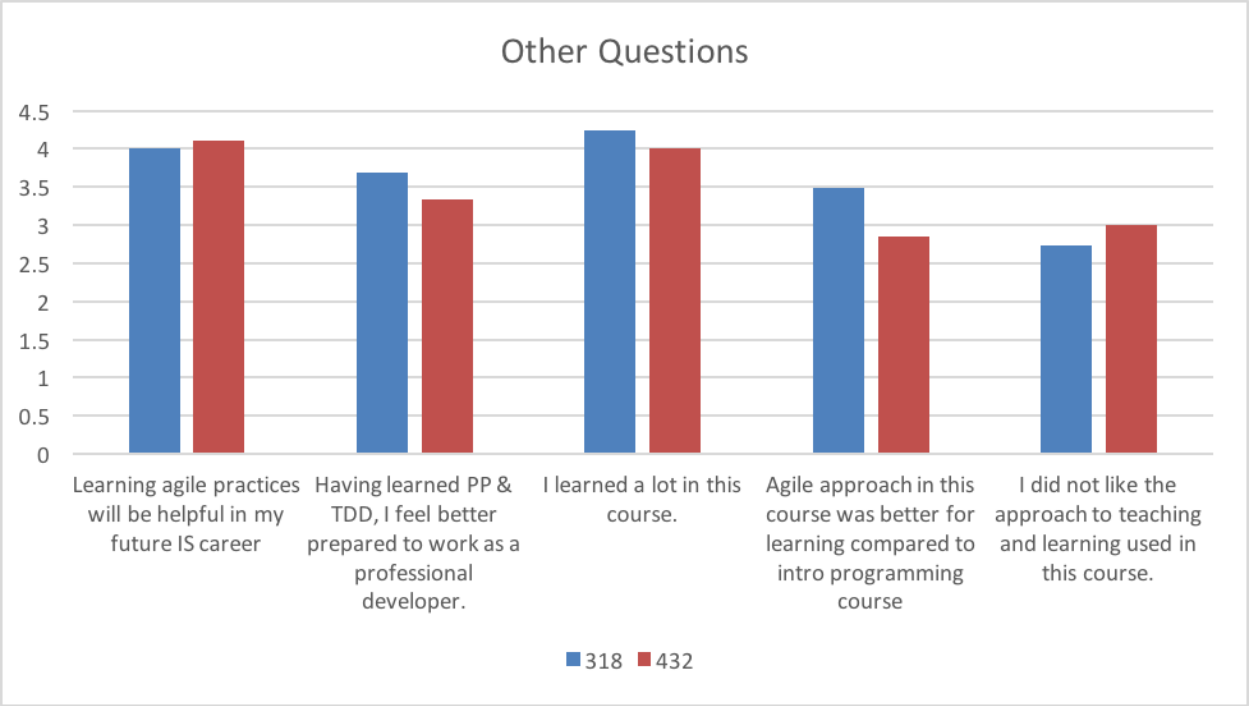

Figure 3. General questions about agile practices in the course.

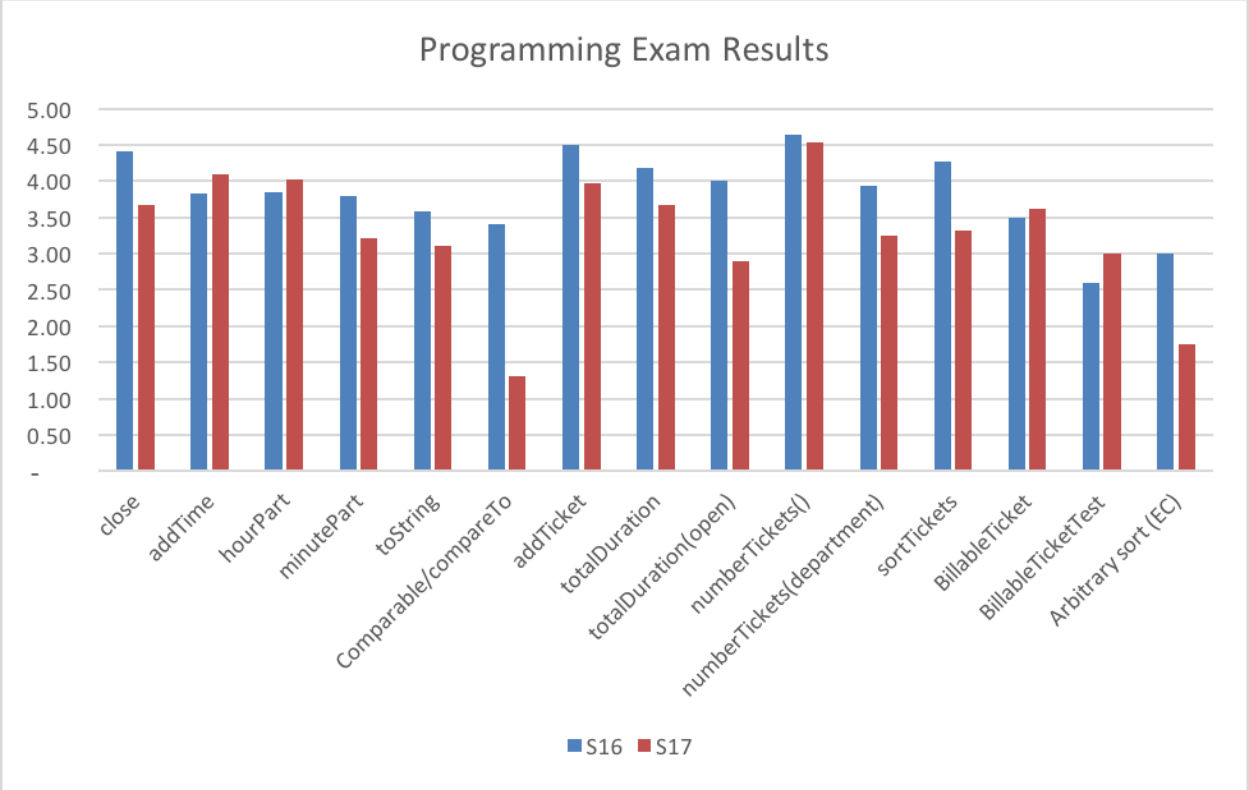

Figure 4. Results of the TDD programming exam for spring 2016 and spring 2017.

6. Discussion

The students who completed both IS 318 and IS 432 took IS 318 a year earlier than the students who only completed IS 318. A few changes were made between the two offerings of the class. The most significant change involved the role of pair programming. In spring 2016, all exercises and assignments were required to be done as a pair to enforce the pair programming practice. However, this requirement was relaxed in spring 2017 for a more nuanced approach where the major assignments were still to be done as a pair, but most of the weekly practice exercises were switched to be done individually to allow each student a better chance of learning the technical material. Many of the exercises were largely tutorials with relatively little independent work, so having two people work on them tended to be redundant.

In the third programming course (IS 432), pair programming and test-driven development was encouraged, but not required. The course revolved around a large group project, so students did naturally collaborate closely, but pair programming was not strictly enforced nor evaluated. Since the main focus of that course is learning web development, there tends to be relatively few parts of the project that lend itself to test-driven development, and it has not been an enforced part of the project. Instead, the focus is on the more managerial aspects of agile, such as using user stories for estimation, sprints, user involvement, and stand-up meetings.

The group of students who went through both courses and had a more negative view of the agile practices, likely did not have that perception as a result of how well they did in IS 318 since their scores on the TDD exam shows that they performed better than the students who only took IS 318. While the change to the pair programming practices likely had a definite impact on perceptions, it would not account for the lower perception of TDD.

While we can't know for sure how these students would have answered the survey immediately after having completed IS 318, it appears likely that their experiences in the year since taking the course had an effect. In that year, they took two other programming classes: IS 432 web development and IS 433 mobile app development. Neither of these classes enforced pair programming nor TDD, so students may have come to see using these practices as less important. They were successful in the subsequent classes without having used the practices, so they may have come to view them as less important. This may have less to do with TDD and pair programming than being much more comfortable with programming by this time. Also, only the students wanting to go into development take the third course.

Most of the previous studies that have looked at using agile practices in order to help students learn programming, have focused on pair programming. This study adds support for using test-driven practices as a way to help students better learn programming practices. In fact, our results indicate that students perceive TDD as even more valuable than pair programming. And the practice has the benefit of being isolated from the social interaction and logistical problems that so often accompany pair programming in a classroom situation.

The lower level of support for pair programming suggests that care must be taken in how pairs are put together and the practice is taught. In IS 318, we used an abbreviated Myers-Briggs test to assign pairs together based on their scores on the Sensing vs. Intuition scale and work ethic. Students would generally prefer to find their own partner, and tend to be upset when they are assigned a partner and things aren't working out. However, the difference in performance between the two student groups here indicates that pair problems don't necessarily translate into lower performance in the class.

There are several consequences that can be drawn from our results. The results show clearly that social aspects of pair programming are very important. Assigning pairs is a critical aspect to this. While there are many ways this can be done, it is clearly an important aspect of the success of pair programming. In our approach, we used a mix of assigning based on personality traits and letting students choose their own partners. Future research could be used to determine the relative efficacy of different ways of assigning pairs. In addition, instructors who use pair programming should also be prepared to spend significant time and effort in working with dysfunctional pairs to ensure they are as successful as possible. One small but significant problem we have encountered is what to do when the class has an uneven number of students. While it can work to have a three-person 'pair' it is far from ideal, as the social issues tend to be magnified with the size of the group, and one person often becomes a 'third wheel' and disengages during collaborative sessions.

It is also clear that in order to ensure students adopt and internalize the practices, they need to be reinforced in subsequent courses. Our results show a clear decline in the perceived value of both practices a year after taking the course. Reinforcing the practices will have the effect of giving students more practice with them, and as they improve as programmers, allow them to see aspects of the practices that weren't obvious to them before.

We plan to repeat the survey for future students to see if the results hold up. In particular, we will be giving the survey to the current group of IS 318 students in one year to see if they show the same decline in opinion about pair programming and test-driven development as we saw with the current group. In addition, we will conduct more focused interviews and focus groups to help us understand the reasons for their responses.

While the results of our investigation suggest an impact of agile practices on student learning, there are limitations. First, and foremost, the study only examined student perceptions. No attempt was made to assess learning outcomes. Future studies could examine both the level of student understanding of the practices, their understanding of the programming language, and the quality of their programming to determine if there is a link between them. Comparison of outcomes could also be done with traditional lecture based courses. Another limitation was the limited sample size. While having 29 responses is somewhat adequate, 29 responses is too limited to generalize the conclusion. Increasing sample size or repeating the survey in future classes could improve the generalization and strength of the conclusions.

References

Bipp, T., Lepper, A., & Schmedding, D. (2008). Pair Programming in Software Development Teams – An Empirical Study of its Benefits. *Information and Software Technology*, *3*(50), 231–240. https://doi.org/10.1016/j.infsof.2007.05.006

Brusilovsky, P., Kouchnirenko, A., Miller, P., & Tomek, I. (1994). Teaching Programming to Novices: A Review of Approaches and Tools. In *Proceedings of ED-MEDIA'94* (pp. 103–110). Vancouver, BC. Retrieved from https://eric.ed.gov/?id=ED388228

Erdogmus, H., Morisio, M., & Torchiano, M. (2005). On the Effectiveness of the Test-first Approach to Programming. *IEEE Transactions on Software Engineering*, *31*(3), 226–237. https://doi.org/10.1109/TSE.2005.37

Faja, S. (2014). Evaluating Effectiveness of Pair Programming as a Teaching Tool in Programming Courses. *Information Systems Education Journal*, *12*(6), 36–45.

Hassan, H. S., Iqbal, N., Sattar, A. R., & Rafi, U. (2015). Scrum Based Quality Enhancement Model for Supervising Final Year Projects of Computer Science. *International Journal of Scientific & Engineering Research*, *6*(9), 384–388.

Hilton, M., & Janzen, D. S. (2012). On Teaching Arrays with Test-driven Learning in WebIDE. In *Proceedings of the 17th ACM Annual Conference on Innovation and Technology in Computer Science Education* (pp. 93–98). New York, NY, USA: ACM. https://doi.org/10.1145/2325296.2325322

Isong, B. (2014). A Methodology for Teaching Computer Programming: first year students' perspective. *International Journal of Modern Education and Computer Science(IJMECS)*, *6*(9), 15.

Janzen, D., & Saiedian, H. (2008). Test-driven Learning in Early Programming Courses. In *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education* (pp. 532–536). New York, NY, USA: ACM. https://doi.org/10.1145/1352135.1352315

Kathuria, V., & Goel, S. (2010). A Novel Approach for Collaborative Pair Programming. *Journal of Information Technology Education: Research*, *9*. Retrieved from https://www.informingscience.org/Publications/1290?Source=% Student Pair Programmers. In *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education* (pp. 7–11). New York, NY, USA: ACM. https://doi.org/10.1145/971300.971307

Keefe, K., Sheard, J., & Dick, M. (2006). Adopting XP Practices for Teaching Object Oriented Programming. In *Proceedings of the 8th Australasian Conference on Computing Education - Volume 52* (pp. 91–100). Darlinghurst, Australia, Australia: Australian Computer Society, Inc. Retrieved from http://dl.acm.org/citation.cfm?id=1151869.1151882

Kumar, G., & Bhatia, P. (2012). Impact of Agile Methodology on Software Development Process. *International Journal of Computer Technology and Electronics Engineering*, *2*(4), 46–50.

Lappalainen, V., Itkonen, J., Isomöttönen, V., & Kollanus, S. (2010). ComTest: A Tool to Impart TDD and Unit Testing to Introductory Level Programming. In *Proceedings of the Fifteenth Annual Conference on Innovation and Technology in Computer Science Education* (pp. 63–67). New York, NY, USA: ACM. https://doi.org/10.1145/1822090.1822110

McDowell, C., Werner, L., Bulock, H., & Fernald, J. (2002). The effects of Pair programming on performance in an introductory programming course. In *Proceedings of the 33rd SIGCSE Technical Symposium on Computer Science Education* (Vol. 34, pp. 38–42). ACM.

Monett, D. (2013). Agile Project-Based Teaching and Learning. In *Proceedings of the International Conference on Software Engineering Research and Practice (SERP)* (p. 1). The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp). Retrieved from http://search.proquest.com/openview/8d19944e2b082656c20b74a648ac13e7/1?pq-origsite=gscholar&cbl=1976341

Mugridge, R. (2003). Challenges in Teaching Test Driven Development. In *Extreme Programming and Agile Processes in Software Engineering* (pp. 410–413). Springer, Berlin, Heidelberg. https:doi.org/10.1007/3-540-44870-5_63

Perera, G. (2009). Impact of using agile practice for student software projects in computer science education. *International Journal of Education and Development Using Information and Communication Technology*, *5*(3), L1.

Rosminah, S., md derus, siti rosminah, & Ali, A. Z. M. (2012). Difficulties in learning programming: Views of students. https://doi.org/10.13140/2.1.1055.7441

Simon, B., & Hanks, B. (2008). First-year Students' Impressions of Pair Programming in CS1. *J. Educ. Resour. Comput.*, *7*(4), 5:1–5:28. https://doi.org/10.1145/1316450.1316455

Spacco, J., & Pugh, W. (2006). Helping Students Appreciate Test-driven Development (TDD). In *Companion to the 21st ACM SIGPLAN Symposium on Object-oriented Programming Systems, Languages, and Applications* (pp. 907–913). New York, NY, USA: ACM. https://doi.org/10.1145/1176617.1176743

Srikanth, H., Williams, L., Wiebe, E., Miller, C., & Balik, S. (2004). On Pair Rotation in the Computer Science Course (pp. 144–149). Presented at the Conference on Software Engineering Education and Training.

Umapathy, K., & Ritzhaupt, A. D. (2017). A Meta-Analysis of Pair-Programming in Computer Programming Courses: Implications for Educational Practice. *ACM Trans. Comput. Educ.*, *17*(4), 16:1–16:13. https://doi.org/10.1145/2996201

Williams, L. (2010). Pair Learning: The Use of Pair Programming in Education. Retrieved June 20, 2017, from http://www.realsearchgroup.org/pairlearning/index.php

Williams, L., Wiebe, E., Yang, K., Ferzli, M., & Miller, C. (2002). In Support of Pair Programming in the Introductory Computer Science Course. *Computer Science Education*, *12*(3), 197–212. https://doi.org/10.1076/csed.12.3.197.8618

Author Biographies

Michael A. Eierman, Ph.D. is a Professor of Information Systems and Chair of the Information Systems Department at the University of Wisconsin Oshkosh College of Business. Dr. Eierman has worked in the information systems field for nearly 30 years as a programmer, analyst, consultant but primarily, as a teacher. His teaching areas include mobile development, project management, systems analysis, and the management of IS. Dr. Eierman's research has taken many directions over his years as a professor but is currently focused on the impact of collaborative and mobile technology.

Jakob H. Iversen, Ph.D. is Professor of Information Systems and serves as Interim Associate Dean of the College of Business at the University of Wisconsin Oshkosh College of Business. He received his M.S. and Ph.D. in Information Systems from Aalborg University, Denmark. His current research interests include software process improvement, agile software development, e-collaboration, and mobile development. He is co-author of Mobile App Development for iOS and Android (Prospect Press). Dr. Iversen teaches and consults on web development, technology innovation, mobile development, information systems management, strategy, and software development processes.