**Trakya Eğitim Dergisi**
Cilt 10, Sayı 1
Ocak 2020, 221-237

*Geliş Tarihi: 11.06.2019*

*ISSN: 2630-6301*
Doi: 10.24315/tred.575098
*Araştırma Makalesi*

**Trakya Journal of Education**
Volume 10, Issue 1
January 2020, 221-237

*Yayına Kabul Tarihi: 12.02.2020*

# The Influence of Pair Programming on Secondary School Students' Confidence and Achievement in Computer Programming

# Eşli Programlamanın Ortaokul Öğrencilerinin Bilgisayar Programlama Özgüven ve Başarısına Etkisi

**Habibe ÇAL[1], Gülfidan CAN[2]**

**Öz:** Bu araştırmada iç içe geçmiş durum çalışması yapılarak eşli programlamanın ortaokul öğrencilerinin bilgisayar programlama özgüven ve başarısına etkisi araştırılmıştır. Beşinci sınıf seviyesinde 35 öğrenci bireysel (n=13) ve eşli (n=22) programlama gruplarına ayrılmış, Scratch programlama etkinlikleri kullanılarak sekiz haftalık bir uygulama yürütülmüştür. Araştırmada nitel veri görüşmelerle, nicel veri ise özgüven anketi ve rubriklerle toplanmıştır. Veri analizi için bağımsız örneklemler t testi ve içerik analizi kullanılmıştır. Uygulama sonunda eşli programlama öğrencilerinin özgüven ve başarısının, bireysel programlama öğrencilerinden daha yüksek olduğu bulunmuştur. Bu çalışma, ortaokul seviyesinde bilgisayar programlama özgüveni ve başarısını artırmak için eşli programlama yönteminin kullanımını desteklemekte, özellikle bilgisayar sayısı yetersiz olan okullara, rekabetçi öğrencilere ve programlamayı yeni öğrenenlere bu yöntemi önermektedir.

***Anahtar sözcükler:*** *Eşli programlama, bilgisayar programlama, özgüven, başarı, ortaokul.*

**Abstract**: The purpose of this embedded case study is to explore the possible influence of pair programming on secondary school students' confidence and achievement in computer programming. A total of 35 students in a fifth-grade class were divided into individual (n=13) and pair programmers (n=22), who then used Scratch programming activities during an eight week implementation. Qualitative data were collected with interviews and quantitative data were collected with a confidence questionnaire and rubrics. Content analysis and independent-samples t tests were conducted for data analysis. The results showed that pair programmers' confidence and achievement for computer programming was higher compared to individual programmers after the implementation. The study supports the use of pair programming in secondary schools, especially where there are limited numbers of computers, competitive students, and novice programmers to increase the confidence and achievement in computer programming.

***Keywords:*** *Pair programming, computer programming, confidence, achievement, secondary school..*

***Cite this article as:***
Çal, H.& Can, G. (2020). The influence of pair programming on secondary school students' confidence and achievement in computer programming. *Trakya Eğitim Dergisi, 10*(1), 221-237.

## UZUN ÖZET

### Giriş

Öğrencilere bir dizi yazılımın kullanımını öğretmek yerine, programlama ile problem çözme aktivitelerinin sağlanması, onların bilişsel olarak daha aktif, sistematik ve araştırmacı olmasına yardımcı olmaktadır. Ancak programlamanın zorunlu olarak müfredata eklenmesi konusunda farklı görüşler vardır. Bilgisayar programlamada başlangıç seviyesinde olan öğrenciler, programlama kavramlarını anlamakta, hatalarını düzeltmekte ve karmaşık programlar yaratmakta zorluk çekmektedirler. Bu sebeple çocukların programlamayı kolayca öğrenebilmesi için Scratch gibi basit ve görsel programlama ortamları oluşturulmuş ve farklı öğretim yöntemleri denenmiştir. Etkili yöntemlerden biri olan eşli programlamada iki öğrenci bir bilgisayarda çalışmakta, biri kodları oluştururken diğeri kodları gözlemleyip eşine yardımcı olmaktadır. K-12 alanında yapılan araştırmalar eşli programlamanın problem çözme ve kritik düşünme becerilerini geliştirdiğini ve programlama

[1] Öğretmen, Yenikent İlksan Ortaokulu, E-posta: habibe_krgll@gmail.com , ORCID: 0000-0001-6365-2020
[2] Dr. Öğr. Üyesi, Orta Doğu Teknik Üniversitesi, E-posta: gcan@metu.edu.tr, ORCID: 0000-0003-0337-4166.

öğrenimini güçlendirdiğini raporlamıştır. Ayrıca, öğrenciler arasındaki etkileşimi ve sosyalleşmeyi artırarak bilgi paylaşımını sağladığı bulunmuştur. Alan yazında eşli programlamanın etkisini araştıran çalışmalar genellikle yetişkinlerle yapılmış ve deneysel yöntemler kullanılmıştır. Ortaokullarda bilgisayar sayısı konusundaki yetersizliklere çözüm olabilecek ve öğrencilerinin programlama eğitimine katkı sağlayabilecek bu yöntemin kullanımı hakkında alan yazında yeterince bilgi bulunmamaktadır. Ayrıca bu yöntemin öğrencilerin bilgisayar programlama özgüveni ve başarısına etkisi konusunda daha fazla araştırmaya ihtiyaç vardır. Bu sebeple bu araştırmanın amacı, eşli programlamanın ortaokul öğrencilerinin bilgisayar programlama özgüven ve başarısına etkisini incelemektedir.

## Yöntem

İç içe geçmiş durum çalışması yapılarak eşli programlamanın etkisi derinlemesine incelenmiştir. Çalışmada nitel veri nicel veri ile desteklenmiştir. Araştırma için ilk yazarın öğretmen olarak çalıştığı, Ankara'da düşük gelir seviyesi olan bir ilçedeki devlet ortaokulunda, bilgisayar programlamayı yeni öğrenen 5. Sınıflar arasından, sınıf mevcudu en düşük olan sınıf seçilmiştir. Sınıftaki 35 öğrencinin yaşları 10 ve 11 arasında değişmektedir. Bu öğrencilerden 19'u kız, 16'sı erkek öğrencidir. Bilişim Teknolojileri ve Yazılım dersinin ilk haftasında öğrenciler bireysel (n=13) ve eşli programlama (n=22) gruplarına ayrılmış, onlara ders ve uygulama hakkında bilgi sağlanmıştır. Sonraki sekiz hafta boyunca ise Scratch web sayfasında bulunan ders planları, etkinlikler ve rubrikler uygulanmıştır. Eşli programlama grubundaki öğrencilerin rolleri her iki haftada bir değiştirilmiştir. İki saatlik dersin ilk saatinde, ders planları ve içinde bulunan etkinliklerden biri kullanılarak düz anlatım yöntemi ile ders yapılmış, öğrencilere etkinlik sırasında yardım ve geri bildirim sağlanmıştır. Dersin ikinci saatinde ise, ilk 10 dakika öğrencilere ikinci etkinlik ve rubrik hakkında bilgi verilmiş ve daha sonra öğrencilerin etkinliği 30 dakika içinde öğretmen desteği olmadan tamamlamaları istenmiştir. Ders sonunda öğretmen öğrencilerin sorularını yanıtlamış ve geri bildirim sağlamıştır. Uygulanan programlama etkinlikleri haftalık olarak rubriklerle değerlendirilmiş ve değerlendirmeler iki kez yapılarak doğruluğu kontrol edilmiştir. Ayrıca, dönem içinde iki kez diğer bir Bilişim Teknolojileri ve Yazılım öğretmeni aynı rubrik ile bağımsız değerlendirme yapmış, iki öğretmenin değerlendirmeleri tutarlı bulunmuştur (ilk uygulama tutarlık=0.82, ikinci uygulama=0.87). Öğrencilerin bilgisayar programlama özgüvenlerini ölçmek amacıyla iki farklı ölçek birleştirilerek oluşturulan bir anket dönem içinde iki kez uygulanmıştır (ilk uygulama Cronbach's Alpha=0.81, ikinci uygulama Cronbach's Alpha =0.88). Öğretmen dönemin son üç haftasında öğrenmeyi pekiştirmek amacıyla etkinliklerle dersi gözden geçirmiş ve problem yaşayan öğrencilere destek sağlamıştır. Bu son üç haftada, öğrenciler bireysel veya eşli programla yapma konusunda serbest bırakılmıştır. Dönem sonunda gönüllü 20 öğrenci ile (7 bireysel, 13 eşli programlama öğrencisi) görüşmeler yapılmıştır. Nicel veri analizi için bağımlı örneklem *t* testi, bağımsız örneklemler *t* testi ve Mann-Whitney *U* testi; nitel veri analizi için ise içerik analizi kullanılmıştır.

## Bulgular ve Tartışma

Nitel analiz sonuçları, eşli programlama kullanılmasının öğrencilerinin bilgisayar programlama özgüven ve başarısını artırdığını göstermiştir. Eşli programlama sırasında öğrencilerin birbirlerine yardımcı olması, bilgi paylaşımı yapması, hatalarını düzelterek problemleri kolayca çözmesi, verilen etkinlikleri hızlı ve kaliteli bir şekilde tamamlaması, onların programlama özgüvenlerini yükseltmiştir. Eşler arasındaki tartışmalar ise özgüvenlerinin düşmesine sebep olmuştur. Benzer şekilde, eşler arasında bilgi paylaşımı, yardımlaşma ve yaratıcılık ile öğrenciler daha doğru kodlar oluşturduklarını, etkinlikleri daha hızlı bitirip ve daha yüksek puanlar aldıklarını belirtmişlerdir. Ayrıca, programlamada özgüven ve başarı arasında güçlü bir bağlantı bulunmuştur. Nicel veri analizleri bu sonuçları desteklemiştir. Bireysel ve eşli programlama öğrencilerinin bilgisayar programlama özgüven değerleri arasında dönem başında önemli bir fark bulunmazken, dönem sonunda eşli programlama öğrencilerinin anlamlı bir farkla bireysel öğrencilere göre daha özgüvenli olduğu bulunmuştur. Aynı şekilde, eşli programlama öğrencilerinin etkinlik puanları bireysel programlama öğrencilerinden anlamlı bir şekilde daha fazladır.

Bulgular, alan yazında K-12 alanında eşli programlama için yapılan az sayıdaki çalışmanın raporladığı olumlu etkiler açısından tutarlıdır. Bireysel ve eşli programla öğrencilerinin aynı ortamda bulunması araştırma sonuçlarını etkilemiş olabilir; ancak bu durum uygulama başında rekabetçi olan ve

bireysel çalışmak isteyen öğrencilerin uygulama sonunda birlikte çalışma ve yardımlaşma tutumlarını geliştirmiştir. Özellikle yetersiz sayıda bilgisayarı olan okullarda öğrenciler halihazırda bir bilgisayarı birlikte kullanmak zorunda kalmaktadırlar. Bilgisayar sayısı yeterli olsa dahi eşli programlama öğrencilerin öğrenme, özgüven ve sosyalleşmesini desteklemesi sebebiyle düzenli olarak kullanılmalı, böylece programlama yaparken öğrencilerin özerkliğinin yanı sıra işbirlikçi tutumunun da gelişmesi sağlanmalıdır. Bu araştırma bir durum araştırması olması sebebiyle sonuçlarının diğer bağlamlara genellemesi sınırlıdır. Ayrıca bu çalışmada eşli programlama öğrencilerinin etkinliklerdeki bireysel performansları ölçülmemiştir. Farklı öğrenci grupları ile çalışmanın tekrarı veya bilgisayar sayısı yeterli okullarda ters çevrilmiş eşli programlamanın incelenmesi faydalı olabilir. Bunu yanında, etkili eşleştirme yöntemleri, eşli programlama ölçme değerlendirme yöntemlerinin araştırılması, öğrencilerin hata bulma ve düzeltme becerilerini geliştirmek için yöntemler ve öğretmen görüşlerinin araştırılması alan yazına katkı sağlayabilir. Bu çalışma, ortaokullarda bilgisayar programlama özgüvenini ve başarısını artırmak için eşli programlama kullanımını desteklemekte, özellikle yetersiz bilgisayar sayısı olan okullara, rekabetçi öğrencilere ve programlamayı yeni öğrenenlere bu yöntemi önermektedir.

## 1. INTRODUCTION

There has been a recent focus on computer fluency rather than computer literacy. Providing problem-solving activities using information technology has been suggested instead of teaching students how to use a list of software (Werner & Denning, 2009). Compared to direct teaching, programming can help children become more cognitively active, systematic, exploratory, and self-directed in solving problems (Maloney, Resnick, Rusk, Silverman, & Eastmond, 2010; Papert, 1980) and it can improve mathematical and social skills (Fessakis, Gouli, & Mavroudi, 2013). As programming gained global popularity in K-12, Turkey has also integrated programming into secondary school curricula, as from 2012.

However, including a new topic into established curricula requires examining students' readiness as well as the required instructional methods, and there are debates ongoing with regards to mandating Computational Thinking into school curricula (Grover & Pea, 2013). Novice programmers in K-12 sometimes experience difficulty with programming concepts, correcting mistakes, or producing complex programs (Denner, Werner, & Ortiz, 2012). One of the strategies suggested to help children learn programming more easily and effectively is the provision of an easy to use programing environment for kids such as LOGO, Scratch, Code.org, and Alice (Fessakis et al., 2013; Grover & Pea, 2013). Several studies reported on the cognitive and affective advantages of using Scratch in schools (Akpinar & Aslan, 2015; Maloney et al., 2010; Wilson & Moffat, 2010; Yünkül, Durak, Çankaya, & Mısırlı, 2017). However, conflicting results have shown that although students' motivation and enjoyment increased with the use of Scratch, there were minimal cognitive improvements realized (Kalelioğlu & Gülbahar, 2014; Wilson & Moffat, 2010). One possible reason for these conflicting results might be due to differences in the instructional methods applied. Introducing a new software for children without the appropriate instructional method does not guarantee the realization of the intended benefits; and it is therefore necessary to examine the effectiveness of instructional methods for programming at the K-12 level (Fessakis et al., 2013).

Consequently, another strategy for facilitating students' learning programming is to use known effective instructional methods such as pair programming. Pair programming is referred as a "modern pedagogical method of teaching" (Nančovska, Kaučič, & Rugelj, 2008, p. 45) or a "teaching-learning strategy" (Mentz, van der Walt, & Goosen, 2008, p. 247). It originated as one of the major practices of Extreme Programming, which differs from traditional software development methods in that it aims to increase the efficiency and the quality of the developed product (Beck, 1999). Pair programming requires programmers to work collaboratively on a task together using a single computer. Each pair has a distinct role. The "Driver" controls the programming environment, creates codes, and tests the codes, while the "Navigator" or "Observer" observes the codes, asks questions, brainstorms, and provides suggestions and corrections (Williams & Upchurch, 2001). These roles should change round on regular basis

to ensure that each pair adequately experiences both roles (Umapathy & Ritzhaupt, 2017; Williams, Wiebe, Yang, Ferzli, & Miller, 2002). Although pair programming is frequently used with adult learners and considered as a collaborative method, it has been suggested that it can also be applied to cooperative learning (Mentz et al., 2008).

Relevant research studies on pair programming have mostly been conducted with adult learners. They have frequently compared individual programming and pair programming, and reported that pair programmers performed more effectively on one or more performance measures including successfully passing programming courses, obtaining higher grades, completing assignments, improving problem-solving and higher-order thinking skills, learning programming, understanding programming concepts, producing quality programs with correct codes and fewer errors, increasing productivity, and faster programming (DeClue, 2003; Dongo, Reed, & O'Hara, 2016; Isong et al., 2016; McChesney, 2016; McDowell, Werner, Bullock, & Fernald, 2006; Nagappan et al., 2003; Nančovska et al., 2008; Salge & Berente, 2016; Umapathy & Ritzhaupt, 2017; Williams & Upchurch, 2001; Williams et al., 2002). However, although pair programmers have been shown to perform better in assignments, there has been no significant difference identified in terms of their exam scores (Nagappan et al., 2003; Williams et al., 2002). It is therefore suggested that each pair programmer learned as much as individual programmers instead of one pair doing all the work (Nagappan et al., 2003).

Collaborative interaction in pair programming can increase adults' confidence in programming (Dongo et al., 2016; McChesney, 2016; McDowell et al., 2006; Williams & Upchurch, 2001), and they can develop more positive feelings and experiences than individual programmers (Dongo et al., 2016; Isong et al., 2016; McChesney, 2016; McDowell et al., 2006; Williams & Upchurch, 2001). Working as pairs can increase effort and motivation (DeClue, 2003; Nagappan et al., 2003) and feel more satisfied having a partner with whom to solve problems through creative and efficient means (Williams & Upchurch, 2001). However, problems with pair working have also been reported due to personality clashes, scheduling, unequally distributed workload, difficulties in communication, and difference in skill levels (DeClue, 2003; McChesney, 2016; Nagappan et al., 2003; Nančovska et al., 2008).

Compared to extensive literature on adult pair programmers, few studies have explored pair programming in schools. For middle school female students using Macromedia's Flash MX, pair programming was found to be an effective method to increase metacognitive activities and enhance problem-solving abilities (Werner & Denning, 2009). By engaging in communication with their pair partner in solving problems, students detected and corrected their errors together and helped each other learn the processes of code debugging (Werner & Denning, 2009). Similar cognitive advantages were also reported for high school students using Delphi programming language; with students seen to question, discuss, and solve problems together through pair programming, their critical thinking was enhanced and programming skills improved (Bailey & Mentz, 2017). In the Turkish literature, Demir and Seferoğlu (2017) found pair programing to be effective in allowing tacit knowledge to be open, and thereby facilitating knowledge transfer among pairs, and increasing efficiency through faster coding with fewer errors.

Pair programming has also been shown to facilitate interaction and socialization among programmers (Bailey & Mentz, 2017). In one study, pair programming for sixth-grade students using Alice helped students to socialize, develop friendships, and increase positive attitudes toward programming (Zhong, Wang, Chen, & Li, 2017). It has also reportedly increased enjoyment in coding (Demir & Seferoğlu, 2017). However, conflicting results have shown that when students work on their own computers and collaborate frequently with others, they complete activities faster as pair programmers and face less conflicts (Lewis, 2011). Gender has not been found to be a statistically significant factor for compatibility in pairs, confidence, or performance; however, females were found to work more harmoniously, were more motivated, and performed better than their male counterparts (Zhong et al., 2017).

Most research reported in the literature have been experimental studies conducted with adults of varying age and programming expertise, while only a few studies have examined K-12 application of this method. Moreover, it is questionable whether or not the available studies were conducted based on the proper implementation of pair programming (Umapathy & Ritzhaupt, 2017). Examining the factors influencing the confidence and achievement of the students using pair programming can help teachers, curriculum developers, and educators to make better informed decisions regarding the use of this method within secondary school programming courses. Also, due to a lack of computers in some schools in Turkey, the use of this method has the potential to alleviate the problem while improving students' attitudes towards working and learning together. Therefore, it is crucial to understand how pair programming influences secondary school students' confidence and achievement in programming through in-depth exploration.

## 2. METHODOLOGY

### 2.1. Research Design

The main research question of this study is "How does the application of pair programming influence the confidence and achievement level of secondary school students in computer programming?" Embedded case research design was chosen for in-depth exploration of the research question within a real environment (Yin, 2003). The collected qualitative data were supported by quantitative data for the purposes of triangulation. Instead of only focusing on pair programmers, the current study intended to also deeply understand both pair and individual programmers' experiences within the same fifth-grade class, and to reveal factors relating to their confidence and achievement.

### 2.2. Context

The context of the study is representative of many urban public secondary schools with limited resources in Turkey. Based on the records of the counseling service in the school where the data were collected for this study, most of the students were from low income families, with parental salaries below the poverty line. In 2017, when the data were collected, the selected school had 1,207 secondary school students with an average class size of 40 students. Each classroom was equipped with a smartboard that included an Internet connection. The school had one computer laboratory that contained one smartboard and 27 computers with an Internet connection. Most of the computers were running on Windows XP operating system, whist a few had Windows 7 with a 2GB memory. Prior to the implementation of this research, the students worked in pairs, but only due to necessity based on the school's limited number of computers.

In Turkey, secondary school refers to students typically aged nine to 15 years old. An "Information Technologies and Software" course is offered to fifth and sixth grade secondary school students as two-hour compulsory course (Millî Eğitim Bakanlığı [Turkish Ministry of National Education], 2018). While in the first semester of the fifth grade, students receive a general introduction to programming, they start actively programming in the second semester. Therefore, the data for the current study were collected during the second semester. The first author of the study was employed as a teacher at the selected school, and the students were familiar with the teacher prior to the implementation. The grading policy for the course included 50% for activities which were a part of this research, 20% for examination, and 30% were project-related.

For the implementation of the current study, Scratch programing environment, lesson plans, and rubrics provided on the Scratch website were all employed. Although all 10 lesson plans were applied during the semester, only the first eight were included in the research study because the ninth lesson plan's activity was relatively short and the 10th was project-based (see Table 1). Each lesson plan typically provided two or three activities. The teacher usually spent the first activity teaching and one of the other activities for implementation. For each lesson plan, the Scratch website also provided example rubrics with five criteria at the three levels (scored as 0 to 2 points). Therefore, the highest achievement score from a rubric was 10 points for each activity. The Scratch website also allowed students to share their projects within their online community; allowing novice programmers to inspect a variety of projects and thereby also provided discussion opportunities.

### 2.3. Teacher's Observations and Reflections Prior to Implementation

One semester prior to the implementation, the teacher observed that most of the fifth grade students had acted selfishly by trying to use certain computers known to work best and attempted to work by themselves even though there were insufficient computers for all of the students in the class. When forced to sit together and work on a single computer, they were mostly noncooperative and did not allow their peers use of the computer in a fair manner, and most complained about the unfair use of the equipment. Most of the students had limited access to computers at home and therefore wanted to use them as much as possible in class. The teacher also observed that the students would frequently compete with each other and did not want to help their classmates. This situation might have been related to the competitive emphasis of the examination system within Turkish K-12 schooling. The teacher believed most of the students were very concerned and motivated about simple grade achievement rather than their actual learning capability or performance.

### 2.4. Participants

One particular school was chosen for this case study as the first author was an appointed teacher for the Information Technologies and Software course. Purposeful sampling strategy was used to select one case among others to reach a rich case and to reveal in-depth information. Fifth-grade students were chosen purposefully because they are considered novice programmers. Among 12 different fifth-grade classes, one class was chosen purposefully as there were fewer students in the class, which enabled the better forming of both pair and individual programmers and allowed for more effective observation of the students during the application. Among the 35 students, 32 had no prior programming experience and the remaining three had only little according to their self-reports.

The students' ages ranged between 10 (40%) and 11 (60%) years old, with an average of $M$ = 10.60. There were 16 (45.7%) male and 19 (54.3%) female students in the class. A confidence questionnaire was administered twice during the implementation to all 35 students. A total of 20 students in the class also volunteered to be interviewed for the study (seven male and 13 female, seven individual programmers and 13 pair programmers), and were considered representative to the class composition in terms of their ages, gender, and achievement scores. In terms of the implementation's weekly activities, four of the students (two pair programmers and two individual programmers) had one week of absenteeism each.

### 2.5. Implementation and Collection of Data

After receiving the approval of the Human Subjects Ethics committee of the Middle East Technical University, the Turkish Ministry of National Education, the principal of the participant school, and of the students and their parents, the research was conducted for a period of eight weeks during the second semester with a total of 35 fifth-grade students. The teacher systematically and randomly organized the students into 11 pair programmers (22 students) and 13 individual programmers by first randomly choosing a number in the classroom list and then skipping two numbers.

During the first week of the semester, the teacher provided the students with an introduction to the course, the computer laboratory rules, the Scratch website, activities, rubrics, and showed the students some programming examples. The teacher then explained about the implementation of the study and announced the list of individual and pair programmers. The teacher also informed the students about the distinct roles of drivers and navigators in pair programming (see Table 1).

**Table 1.** Implementation and data collection process

| Week | Implementation | Rubrics | Application | Other Data Collection |
|---|---|---|---|---|
| Week 1 | Information | | | |
| Week 2 | Lesson Plan 1 | Rubric 1 | Applied | |
| Week 3 | Lesson Plan 2 | Rubric 2 | Applied | Interrater reliability |
| Week 4 | Lesson Plan 3 | Rubric 3 | Applied | Confidence questionnaire |
| Week 5 | Lesson Plan 4 | Rubric 4 | Applied | |
| Week 6 | Lesson Plan 5 | Rubric 5 | Applied | |

| | | | | |
|---|---|---|---|---|
| Week 7 | Lesson Plan 6 | Rubric 6 | Applied | |
| Week 9 | Lesson Plan 7 | Rubric 7 | Applied | Inter-rater reliability |
| Week 10 | Lesson Plan 8 | Rubric 8 | Applied | Confidence questionnaire |
| Week 11 | Activity, Review, and Feedback | | Students' choice | Interviews |
| Week 12 | Activity, Review, and Feedback | | Students' choice | Interviews |
| Week 13 | Summary and Game | | Students' choice | |

Throughout the implementation, for the first 40-minute sessions of each class, the teacher provided a lesson by way of the direct instruction method utilizing Scratch lesson plans. First, the teacher stated the objectives of the lesson, then explained and showed the first activity step-by-step using the classroom smartboard. As the students applied the activity on their respective computers, the teacher provided them with the necessary help, guidance, and feedback. During the second session, for the first 10 minutes of each lesson, the teacher provided information about the activity and its rubric on the smartboard, as well as providing directions and guidance to the class. Both groups were asked to complete the activity within a period of 30 minutes, and without the teacher's help. If a student became stuck, they were permitted to use the available resources on the Scratch website. Students who completed the activities before the end of the 30-minute period were asked to continue exploring the Scratch programming environment. Undertaking any activities on the computers that were irrelevant to the assigned programming activity were not permitted. During the period when the students were working, the teacher only observed them. At the end of the 30-minute session, the teacher then answered the students' questions and provided feedback on their work. During a 15-minute break that followed, the teacher also evaluated the task performance using the rubrics from the Scratch website. Within one week and prior to the next scheduled class session, the teacher re-evaluated the students' activities to assure the accuracy of the scores recorded.

Pair programmers were asked to stay in the same role for periods of two weeks to adapt to each role. The teacher kept track of switching the roles. Throughout the implementation, the teacher made sure that each student in pair programming contributed to each activity.

The teacher administered a programming confidence questionnaire twice; once at the end of the forth week and again at the end of the 10$^{th}$ week. Since the students had no prior experience in completing a Likert-type instrument, the teacher explained how the students should complete the questionnaire and explained the importance for applying their honesty during each administration. The students completed the questionnaire within a period of 30 minutes. The necessary permissions were taken from the school for this level of duration. During the third and the ninth weeks, a second teacher who gave the same course to a different class evaluated the participant students' performances using the same rubric for the purposes of interrater reliability.

For the 11$^{th}$ and 12$^{th}$ weeks of the semester, the students were permitted to select whether to work as individual programmer or pair programmer. The teacher used the activities to reinforce learning during these two weeks of the course and provided feedback to those students who had experienced difficulties. Interviews were also conducted during the same two weeks at the end of the lesson, and were audio-recorded using a digital recording device with the permission of the students. It was observed that the students exhibited no discomfort due to the presence of the recording device. The interviews were conducted individually in the computer laboratory or in the school's library. To make the students feel comfortable, a series of warm-up questions were asked by the interviewer. The interviews each lasted for an average of $M$=6.37 ($SD$=1.50) minutes. During the final week of the course, the teacher provided a review and recap lesson and permitted the students to play games that they had created themselves using Scratch.

### 2.6. Instruments

For this case study research, several types of data sources were employed including audio-recordings of the students' interviews, results from a twice administered confidence questionnaire, and the students' achievement scores based on rubrics on the Scratch website.

***Confidence questionnaire:*** Due to nonexistence of a comprehensive scale to measure student programming confidence, items from two different scales were combined. From the "Computer Science Attitude Survey" using five-point, Likert-type items, 11 of the items were selected to measure the students' confidence in learning computer science and computer programming (Wiebe, Williams, Yang, & Miller, 2003). The Cronbach's Alpha internal consistency coefficient was found to be .91 for the

scale (Wiebe, Williams, Yang, & Miller, 2003). Minor revisions were applied to some of the selected scale items in terms of changing the phrase "computer science" to "programming".

The second scale that was used was "The TIMSS 2011 Students Confident in Mathematics Scale" and included nine items aimed at eight-grade students (Martin & Mullis, 2012). The scale's Cronbach's Alpha coefficient was found to be .87 for the context of Turkey (Martin & Mullis, 2012). Due to the relationship between mathematics and programming (Papert, 1980), all nine of the scale's items were included in the confidence questionnaire created for application within the current study, with minor changes applied by substituting the word "Programming" for "Mathematics".

As both of the aforementioned scales were developed for the English Language, items of the newly creatd confidence questionnaire were translated into Turkish by the researchers of this study, and then the translations reviewed by two English language experts familiar with the subject of programming. The translated confidence questionnaire was then tested with four additional students with similar educational levels and backgrounds as the participants of the study. Using the think-aloud procedure, the four reviewing students provied feedback about the questionnaire items' clarity. After content validity review of the final questionnaire with two faculty members from the Computer Education and Instructional Technology (CEIT) program, the items were approved as sufficiently representative to measure secondary school students' programming confidence level. The Cronbach Alpha coefficient of the final 20-item questionnaire was found to be .81 for the first implementation and .88 for the second implementation.

***Interview Protocol:*** A semi-structured interview protocol with five primary questions was developed by the researchers of the current study to explore the participant students' opinions regarding their experiences, perceived confidence and achievement in programming throughout the implementation. To test the clarity of the questions, four students from the sixth grade at the same participant school were interviewed. After the interview protocol was revised for its clarity, bias, and the target students' age level, it was further examined in terms of its content validity by a computer teacher and two faculty members from the CEIT department.

***Rubrics:*** The example rubrics were taken from the Scratch website and slightly revised in accordance with the activities selected for this study's implementation, and were then examined by two content experts from the CEIT department. The rubrics were subsequently translated into the Turkish language and approved by two language experts familiar with programming. The two computer teachers who conducted the interrater reliability for the current study then reviewed and discussed the rubric items prior to implementation to ensure they understood the same criteria while evaluating the students' activities. The two teachers' scores were found to be consistent for both application (82% and 87% consistency).

## 2.7. Data Analysis

For the analysis of the interviews, the audio recordings of the interviews were transcribed verbatim, and then analyzed according to themed content analysis (Yıldırım & Şimşek, 2016). Codes were created, classified, and then themes developed, organized, and defined. Among the 20 interviews that were conducted, two of the interviews (10%) were coded independently by two coders and their categories compared and combined through mutual discussion. Data saturation has deemed to have been achieved following analysis of about half the interview transcripts. For the analysis of the quantitative data, descriptive statistics, independent-samples *t*-test, and Mann-Whitney *U* Test were conducted.

## 2.8. Trustworthiness

In terms of assessing the study's credibility (Lincoln & Guba, 1985) the implementation duration lasted for a period of eight weeks, and a variety of data were collected for the purposes of triangulation. Two researchers from the CEIT department and one computer teacher evaluated and discussed the research and provided feedback to the researchers. For transferability, thick descriptions about the context and the implementation were used in reporting the study. For dependability, interrater reliability assessment was conducted. For confirmability, the teacher maintained a diary throughout the implementation semester to record all the process in detail and in a reflexive manner. A variety of data were collected for

the confirmation of the study's results, and the whole research process was reported in detail. Throughout the research process, the teacher attempted to control any self-bias biases and encouraged the participant students to provide honest responses.

## 3. QUALITATIVE DATA ANALYSIS RESULTS

### 3.1. Factors Influenced Programming Confidence

Both groups attributed their increased or decreased confidence to similar factors. Pair programmers frequently reported the advantages of cooperation, while individual programmers attributed their decreased confidence to being unsupported when they experienced difficulties. The pair programmers reported that when they encountered problems during programming, being in pairs increased their confidence (see Table 2). Their confidence increased toward programming when they helped each other to complete the activities faster and more effectively, they shared knowledge, corrected their mistakes, and found solutions easily together through brainstorming during the problem-solving process. For individual programmers on the other hand, finding solutions through a series of trial and error were reported to be difficult and that they often could not spot their own mistakes. They felt that they needed help from their teacher, classmates, or other resources to complete the activity.

**Table 2.** Factors influenced programming confidence

| **Pair programmer confidence** | **Pair *n*** | **Individual programmer confidence** | **Indiv. *n*** | **Total** |
|---|---|---|---|---|
| **A. Problem Solving Process** | | **A. Problem Solving Process** | | |
| Finding solutions (easy) | 8 | Finding solutions (hard) | 7 | 15 |
| Helping each other | 13 | Helping each other | 0 | 13 |
| Sharing knowledge | 10 | Sharing knowledge | 0 | 10 |
| Correcting mistakes | 7 | Correcting mistakes | 0 | 7 |
| Knowledge source for problem solving | 0 | Knowledge source for problem solving | 5 | 5 |
| **B. Programming Process** | | **B. Programming Process** | | |
| Task completion time (fast) | 7 | Task completion time (slow) | 7 | 14 |
| Learning programming (high) | 9 | Learning programming (low) | 4 | 13 |
| Quality of product (high) | 4 | Quality of product (low) | 5 | 9 |
| Motivation (high) | 5 | Motivation (low) | 3 | 8 |
| **C. Being in Pair or Individual** | | **C. Being in Pair or Individual** | | |
| Programming ability differences (good) | 7 | Programming ability differences | 0 | 7 |
| Heavy workload | 0 | Heavy workload | 7 | 7 |
| Disagreements (high) | 6 | Disagreements (low) | 0 | 6 |

In terms of programming process, the pair programmers' confidence in programming increased as they learned programming, increased in motivation, and completed their assigned activities both quickly and to a high quality. Similarly, when individual programmers completed the activities very slowly or not at all within the class duration, when they felt they could not learn much about programming, when they produced low quality products, or when they felt unmotivated due to these sorts of difficulties, their confidence dropped as a result. Pair programmers reported that having a more knowledgeable pair partner made them feel more confident. However, disagreements between pairs negatively influenced their confidence. For individual programmers, their confidence dropped as they felt overwhelmed with the workload.

> *My friend contributed to me in programming. With my friend we completed our coding quicker and faster… I was able to ask my friend when I made a mistake. Even in difficult work we believed we completed our work quicker together, and so we believed in ourselves.* (Pair programmer)

> *At the beginning of the programming course I had some self-confidence. And it made me happy to sit at the computer by myself and use the computer. But once I did coding in Scratch, I felt a lack of confidence toward programming. When I encountered problems, I had difficulty with codes to solve them… I tried to solve by trial and error; some worked, but some didn't… As the coding got harder, my success dropped… When I couldn't find the solutions, my confidence dropped too.* (Individual programmer)

### 3.2. Factors Influenced Programming Achievement

According to the pair programmers, sharing knowledge, getting help from their pair partners, testing codes together, and being creative by brainstorming with their partner were methods that contributed to achievement (see Table 3). Access to resources were important for individual programmers' success. In terms of achievement, the students frequently reported the importance of working codes, grades, completion duration, and required effort.

Individual programmers reported that the required effort for success was higher for them compared to pair programmers who completed activities relatively faster, with fewer coding errors, and therefore received higher grades. Individual programmers reported having several coding errors, completing the activities very slowly, and thereby receiving lower grades.

The interview results showed that the students' confidence and achievement were closely linked, and that the students' emotional state also played a role. Pair programmers frequently reported positive emotions such as feeling confident, relaxed, productive, motivated, friendly, and having fun. However, the emotions stated by the individual programmers were mostly negative, including feeling diffident, panicked, unproductive, unmotivated, isolated, frightened, and even desperate.

> *The coding started to get harder, and everything I did turned out wrong… I couldn't maintain my focus and my mind would drift. I couldn't solve the problems when I was alone… My self-confidence dropped. I had a hard time as I made mistakes. I wished I'd worked with my friends to help me find my mistakes by talking to them. I would have had more self-confidence and be more successful if my friend could have told me what to do.* (Individual programmer)

> *While we were working together with my friend, we did better by combining our knowledge. I added the codes, and my friend checked the accuracy of the codes… My friend corrected my mistakes and checked my work. This made us more successful as a team.* (Pair programmer)

**Table 3.** Factors influenced programming achievement

| Pair programmer achievement | Pair *n* | Individual programmer achievement | Indiv. *n* | Total |
|---|---|---|---|---|
| **A. Method used for Achievement** | | **A. Method used for Achievement** | | |
| Knowledge sharing | 13 | Knowledge sharing | 0 | 13 |
| Getting help | 13 | Getting help | 0 | 13 |
| Testing codes together | 8 | Testing codes together | 0 | 8 |
| Access to resources | 0 | Access to resources | 5 | 5 |
| Being creative | 4 | Being creative | 0 | 4 |
| **B. Programming Process** | | **B. Programming Process** | | |
| Amount of coding errors (low) | 10 | Amount of coding errors (high) | 6 | 16 |
| Activity completion duration (fast) | 7 | Activity completion duration (slow) | 5 | 12 |
| Grades for activities (high) | 7 | Grades for activities (low) | 5 | 12 |
| Required effort to achieve (low) | 0 | Required effort to achieve (high) | 6 | 6 |

### 3.3. Final Observations and Reflections of the Teacher

The teacher observed that the students liked the Scratch programing interface and enjoyed working with it during the implementation. Compared to the previous semester prior to the implementation, most of the students' interviews revealed a significant change of attitude from their working individually to collaboratively. Instead of their previous attempt to utilize the few computers for themselves, they wanted instead to work in pairs, both for their achievement and also for their self-confidence. The teacher also observed that even though some of the individual programmers experienced difficulties, they continued to work hard to complete all of the activities instead of just giving up. The students' competitive attitude was seen to continue to be exhibited. In each lesson, the students still competed to achieve the best

score and to have the fastest task completion time. The teacher still felt that the students were focused more on achieving their course grades than in the learning of programming, and that the students' confidence was still linked to realizing high grades from the course.

## 4. QUANTITATIVE DATA ANALYSIS RESULTS

To support the qualitative data of the current study, quantitative data were also collected. The data were analyzed using IBM SPSS v22.0 statistical analysis software. The data were prepared for quantitative analysis and negative items were reverse-coded. There were no missing data found in the confidence questionnaires returned, but there were four instances of missing values in the achievement data due to four students' absenteeism (two pair programmers and two individual programmers). These missing values were replaced with the sample mean, for the sake of data completeness. Normality assumption was met for the confidence questionnaire data and for the individual programmers' achievement data, but not for the pair programmers' achievement data based on Shapiro-Wilk test, Q-Q plots, and histograms. Both *t*-tests and nonparametric procedures were applied and the results compared as the sample sizes between two groups were not equal and the sample size considered to be small.

### 4.1. Programming Confidence

The differences in mean scores in the confidence questionnaire data between individual programmers and pair programmers were analyzed for both applications of the questionnaire using independent-samples *t*-test and Mann-Whitney *U* Test. Independent-samples *t*-test results showed that, for the first application of the confidence questionnaire there was no significant difference found between the scores of the individual and pair programmers ($t(33) = .13$, $p = .90$) (see Table 5). In the second application, however, there were significant differences found between the mean scores of the individual and pair programmers ($t(33) = -2.76$, $p < .05$), with a large effect size (eta square = .19). Mann-Whitney *U* Test confirmed these results (see Table 9).

**Table 5.** Independent-samples *t*-test results for confidence questionnaire data

| | | ***M (SD)*** | ***M (SD)*** | **Levene's Test for Eq. of Variances** | | ***t*-test for Equality of Means** | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | ***Individ.*** | ***Pair*** | ***F*** | **Sig.** | ***t*** | ***df*** | **Sig. (2-tailed)** | ***M* Diff.** |
| Confidence Questionnaire Application 1 | Equal variances assumed | 3.13 (.97) | 3.11 (1.04) | .07 | .80 | .13 | 33.00 | .90 | .02 |
| Confidence Questionnaire Application 2 | Equal variances assumed | 2.93 (.58) | 3.29 (1.03) | .14 | .71 | -2.76 | 33.00 | .01 | -.36 |

Descriptive analysis of the confidence questionnaire data is presented in Table 6. As can be seen, the participants' ratings mostly showed positive confidence scores. They were most confident about learning programming and performing well on the programming course. Their confidence ratings for advanced programming or difficult programming problems were also shown to be positive.

**Table 6.** Confidence questionnaire results for two implementations

| | | Individual programmers | | Pair programmers | |
|---|---|---|---|---|---|
| | | Confid. 1 *M (SD)* | Confid. 2 *M (SD)* | Confid. 1 *M (SD)* | Confid. 2 *M (SD)* |
| 1. | I am sure that I could do advanced work in computer science. | 4.15 (.89) | 3.30 (.85) | 3.86 (.71) | 4.40 (.66) |
| 2. | I am sure that I can learn programming. | 4.61 (.76) | 3.76 (.92) | 4.54 (.85) | 4.63 (.58) |
| 3. | I think I could handle more difficult programming problems. | 3.84 (1.14) | 3.07 (1.18) | 3.68 (.94) | 3.90 (.75) |
| 4. | I can get good grades in programming course. | 4.15 (1.06) | 3.61 (.65) | 4.04 (.84) | 4.54 (.50) |
| 5. | I have a lot of self-confidence when it comes to programming. | 4.23 (1.01) | 3.76 (.92) | 4.40 (.73) | 4.27 (.93) |
| 6. | I am no good at programming. | 1.53 (.66) | 2.30 (1.18) | 1.95 (1.04) | 2.31 (1.46) |
| 7. | I do not think I could do advanced programming. | 3.15 (1.28) | 2.84 (1.21) | 2.63 (1.17) | 2.36 (1.09) |
| 8. | I am not the type to do well in computer programming. | 2.69 (1.37) | 2.23 (1.36) | 1.90 (1.01) | 2.22 (1.23) |
| 9. | For some reason even though I work hard at it, programming seems unusually hard for me. | 2.00 (1.35) | 2.30 (1.10) | 2.36 (1.49) | 2.13 (1.32) |
| 10. | Most subjects I can handle O.K., but I have a knack for flubbing up programming problems. | 3.07 (1.65) | 3.38 (1.19) | 2.50 (1.30) | 2.36 (1.36) |
| 11. | Programming has been my worst subject. | 1.84 (1.28) | 2.15 (1.06) | 1.77 (1.19) | 2.22 (1.30) |
| 12. | I usually do well in programming. | 3.69 (1.25) | 3.38 (1.32) | 4.36 (.90) | 4.22 (1.19) |
| 13. | Programming is more difficult for me than for many of my classmates. | 2.46 (1.45) | 2.30 (1.03) | 2.27 (1.12) | 2.50 (1.14) |
| 14. | Programming is not one of my strengths. | 2.30 (1.31) | 2.38 (.96) | 2.13 (.94) | 2.50 (1.62) |
| 15. | I learn things quickly in programming. | 3.69 (1.54) | 3.46 (1.19) | 4.09 (1.01) | 4.22 (.75) |
| 16. | Programming makes me confused and nervous. | 2.15 (1.06) | 2.23 (.83) | 1.81 (.95) | 2.36 (1.25) |
| 17. | I am good at working out difficult programming problems. | 3.53 (1.12) | 3.00 (1.15) | 3.63 (1.00) | 4.09 (1.06) |
| 18. | My teacher thinks I can do well in programming lessons with difficult materials. | 4.15 (.89) | 3.38 (.86) | 4.27 (.82) | 4.31 (.94) |
| 19. | My teacher tells me I am good at programming. | 3.61 (1.12) | 3.30 (.63) | 3.95 (.95) | 4.18 (1.05) |
| 20. | Programming is harder for me than any other subject. | 1.69 (.94) | 2.38 (.86) | 2.04 (1.43) | 2.00 (1.19) |
| **Average** | | **3.13 (.97)** | **2.93 (.58)** | **3.11 (1.04)** | **3.29 (1.03)** |

***Note:*** *1. Strongly Disagree, 2. Disagree, 3. Neutral, 4. Agree, 5. Strongly Agree.*

## 4.2. Programming Achievement

The individual programmers' mean scores for achievement data ranged between $M$ = 6.23 ($SD$ = 1.87) and $M$ = 9.46 ($SD$ = .77). For the pair programmers, the mean scores for achievement data ranged between $M$ = 9.81 ($SD$ = .39) and $M$ = 9.36 ($SD$ = 1.25) (see Table 7). While the pair programmers' scores fluctuated around a score of 9, the individual programmers scores were around 8 and then dropped towards the end of the semester. The main difference between the mean scores of the two groups was observed for Activity 7, which required the students to design a game.

**Table 7.** Descriptive statistics for achievement

| Activity | Act 1 | Act 2 | Act 3 | Act 4 | Act 5 | Act 6 | Act 7 | Act 8 | Total |
|---|---|---|---|---|---|---|---|---|---|
| Indiv. programmers | 7.92 | 8.53 | 8.00 | 9.46 | 8.53 | 7.76 | 6.23 | 7.69 | 63.38 |
| *M (SD)* | (1.38) | (1.61) | (1.52) | (.77) | (.96) | (1.53) | (1.87) | (1.31) | (6.47) |
| Pair programmers | 9.72 | 9.81 | 9.63 | 9.36 | 9.45 | 9.72 | 9.54 | 9.54 | 76.27 |
| *M (SD)* | (.45) | (.39) | (.65) | (1.25) | (1.01) | (.45) | (.80) | (.50) | (4.33) |
| Mean difference | 1.80 | 1.28 | 1.63 | -.10 | .92 | 1.96 | 3.31 | 1.85 | 12.89 |

Homogeneity of variance assumption was not met based on Levene's Test for the achievement rubric ($F$ = 6.72, and $p$ = .01) (see Table 8). Independent-samples $t$-test showed that significant difference ($t$(18.45) = -6.38, $p$ = .00) was found between the individual programmers' achievement mean scores ($M$ = 63.38, $SD$ = 6.47) and pair programmers' achievement mean scores ($M$ = 76.27, $SD$ = 4.33), with a large effect size (eta square= .55). Mann-Whitney $U$ Test confirmed these results (see Table 9).

**Table 8.** Independent-samples $t$-test results for achievement scores

| | **Levene's Test for Eq. of Variances** | | ***t*-test for Equality of Means** | | | |
|---|---|---|---|---|---|---|
| | ***F*** | **Sig.** | ***t*** | ***df*** | **Sig. (2-tailed)** | ***M* Diff.** |
| Equal variances assumed | 6.72 | .014 | -7.06 | 33 | .00 | -12.88 |
| Equal variances not assumed | | | -6.38 | 18.45 | .00 | -12.88 |

**Table 9.** Mann-Whitney $U$ Test for programming confidence and achievement

| | | ***n*** | ***M* Rank** | **Sum of Ranks** | ***z*** | ***p*** | **Conclusion** | ***t*-test** |
|---|---|---|---|---|---|---|---|---|
| Questionnaire Application 1 | individual | 13 | 18.96 | 246.50 | -.428 | .67 | Not significant | Agreed |
| | pair | 22 | 17.43 | 383.50 | | | | |
| Questionnaire Application 2 | individual | 13 | 12.12 | 157.50 | -2.624 | .01 | Significant | Agreed |
| | pair | 22 | 21.48 | 472.50 | | | | |
| Achievement Rubric Scores | individual | 13 | 8.12 | 105.50 | -4.428 | .00 | Significant | Agreed |
| | pair | 22 | 23.84 | 524.50 | | | | |

## 5. DISCUSSION AND CONCLUSION

In this study of fifth-grade secondary school students who worked as pair programmers or individual programmers for an eight-week implementation with Scratch, the pair programmers showed higher levels of confidence and achievement compared to the individual programmers. Quantitative and qualitative data analysis results both showed that while the students' confidence toward programming was not found to be significantly different between the two groups at the beginning of the semester, as the semester progressed and the programming activities became more complicated, the individual programmers started to lose confidence, while the pair programmers gained in confidence. In terms of achievement, the pair programmers received relatively higher scores than the individual programmers.

The results of the current study were consistent with the other limited number of studies in the literature, in that pair programming positively affected increased confidence in programming (Dongo et al., 2016; McChesney, 2016; McDowell et al., 2006; Williams & Upchurch, 2001). The main factors that increased the pair programmers' confidence in programming in the current study were solving problems easily, helping each other, sharing knowledge, correcting each other's mistakes, completing the activities quickly and to a good quality, increased learning, increased motivation, and working with a peer who had a higher programming ability. Similar to most studies in the literature based on K-12 students, pair programmers' achievement in programming was also positively influenced by pair programming. Their achievement increased when they shared knowledge, received help, tested codes together, were creative, had fewer coding errors, and completed tasks quickly (Bailey & Mentz, 2017; Demir & Seferoğlu, 2017; Werner & Denning, 2009). Therefore, this current study supports that pair programming is an effective method to increase the confidence and achievement of secondary school students in computer programming.

Having pair programmers and individual programmers working alongside each other in the same learning environment might have influenced the validity of the current study's results. The pair programmers' increased confidence and achievement might also have negatively influenced the individual programmers' confidence and achievement. However, the reverse scenario may also have been possible if the individual programmers had outperformed

the pair programmers. As seen in the literature, pair programmers did not always report better performance over individual programmers (Salge & Berente, 2016). However, as an advantage of having both pair and individual programmers working alongside each other in the same environment, almost all of the students in the current study showed a preference for working cooperatively for the remainder of the semester following the implementation, while they were observed to be highly individualistic during the previous semester. This suggests that, for a competitive group of students with limited available educational resources, pair programing can help students to develop more positive attitudes toward cooperation, sharing, and learning from each other. If young students' preference for competition needs to be satisfied, competing teams instead of competing individually is also an option (Denner et al., 2012; Fessakis et al., 2013) to be considered at the design stage of future implementations.

The interview results showed a strong mutual interaction between the confidence and achievement levels of the students for programming. The students in this study reported that they could be successful by attentively listening to the first lesson and then reviewing at home; however, they mostly preferred to work in pairs because they considered the presence of a peer to increase their levels of confidence and achievement (Dongo et al., 2016; McChesney, 2016; McDowell et al., 2006; Williams & Upchurch, 2001). This suggests that pair programming can be beneficial for novice programmers in secondary schools to increase their confidence in learning programming during their programming education.

The application of pair programming can be arranged based on the number of available computers in the classroom or laboratory environment. It was reported in the literature that when each student worked on their own computers and collaborate frequently, they completed the activities faster than pair programmers who worked together throughout each task (Lewis, 2011). If there are adequate numbers of computers for each student in the class, another alternative is to use inverted pair programming, in which pairs design the program together, then split to work individually during the implementation, and then come together again as a pair for the testing stage (Swamidurai & Umphress, 2015). However, when there is an inadequate number of computers, the students may have to sit in pairs just from a numerical and practical perspective (Demir & Seferoğlu, 2017). Therefore, it is pedagogically more advantageous to use pair programming by applying the procedures and practices to make this method more effective, than simply leaving the collaboration between pairs to chance. The formation can be regularly changed between pair programming and individual programming to provide students with both experiences of working autonomously and collaboratively. So as to make sure each student in the pairs adequately learns the prescribed level of programming, they can be asked to undertake a similar activity on their own after the application of pair programming.

The matching of pairs needs to be addressed with due care while applying pair programming. In the literature, the problems reported for adult pair programmers such as scheduling, unequal workload, differences in skill levels, or difficulties in communication (DeClue, 2003; Isong et al., 2016; McChesney, 2016; Nagappan et al., 2003; Nančovska et al., 2008) were not observed in the current study. Nančovska et al. (2008) reported that most of the disadvantages associated with pair programming related to the professional software environment may not appear as they are largely irrelevant to the educational setting. As the students in the current study were all novice programmers enrolled at the same secondary grade, there was not much difference in terms of their programming expertise or age level. The disagreements they did have were mostly minimal and related to the design of the program instead of the actual coding. Although pair programming can facilitate positive emotions (Demir & Seferoğlu, 2017; Zhong et al., 2017), disagreements in the pairs had a decreasing effect on their confidence. In the current study, the students' interviews did not show any indication that gender was an important factor (Zhong et al., 2017). However, gender composition of programming pairs could also be examined at the K-12 level.

In the current study, as the activities became more difficult, both groups started to make more mistakes and experienced difficulties in performing code debugging. However, the pair programmers were able to correct more of their mistakes, whilst the individual programmers sometimes struggled to complete the activities. It is possible, however, that the students' debugging capabilities in both groups were low due to their being novice programmers.

Therefore, it is advisable to teach students debugging strategies prior to the implementation process and to provide them with more extensive resources throughout the implementation.

In order for pair programming to be successful, it is important that students fully understand the roles and the procedures. Therefore, aside from teachers' presentations about these roles and procedures, a small pamphlet or handbook for pair programming could be provided to the participating students (Zhong et al., 2017). Similarly, for individual programmers, a variety of resources and step-by-step self-learning materials could be provided. The assessment and evaluation methods for pair programming should also be examined in future studies for secondary school students (Williams et al., 2002).

In the current study, the students mostly focused on task completion duration. This was likely due to the tasks needing to be completed within a 30-minute deadline. However, spending longer periods on programming may increase students' understanding and thereby improve the quality of the developed product (Salge & Berente, 2016). With pair-explanations, code reviews, and reflections, students can learn better during pair programming (Williams & Upchurch, 2001). However, within a limited class time it may not be feasible; teachers may still attempt to concentrate more on motivating students through increased reflection and pair-explanation rather than simply targeting students to complete the activities as quickly as possible.

Pair programming has been associated with problems related to increased noise levels in computer laboratories (Isong et al., 2016). However, on the positive side, pair programming can help teachers to manage lessons better with peers helping each other (Nagappan et al., 2003). With simple questions being handled within pairs, the teacher can use their time better in addressing the more significant questions (Nagappan et al., 2003). Moreover, with pair programming, fewer assignments need to be evaluated by the teacher, instances of cheating being reduced (Williams & Upchurch, 2001), and teacher stress diminished as a result (Williams et al., 2002). Therefore, the current study supports the use of pair programming to help teachers as well as their students. Exploring teachers' perspectives on pair programming can be valuable.

The current research was designed and conducted as a case study, and therefore the results may not be generalizable to other contexts or student groups. One limitation of the current study is that pair programmers' individual performances were not measured. The study instead attempted to reveal any differences in achievement and confidence scores between two distinct groups. Moreover, due to the age level of the students, the participants of the current study did not provide in-depth explanations during their interviews. The application of different research designs for the purposes of generalization could help future studies to be comparable to the current study. Moreover, the use of pair programming with other programming environments, different participant age groups, and different expertise level students in other contexts could provide a valuable addition to the literature. Compared to pair programming with adult software developers, whose focus is largely on software quality, cost, and development duration, the current study contributes to the literature by providing an educator's perspective that could guide further studies in applying this method to other K-12 educational contexts.

## 6. REFERENCES

Akpinar, Y., & Aslan, U. (2015). Supporting children's learning of probability through video game programming. *Journal of Educational Computing Research, 53*(2), 228-259. doi:10.1177/0735633115598492

Bailey, R., & Mentz, E. (2017). The value of pair programming in the IT classroom. *Independent Journal of Teaching and Learning, 12*(1), 90-103.

Beck, K. (1999). Embracing change with extreme programming. *Computer, 32*(10), 70-77. doi:10.1109/2.796139

DeClue, T. H. (2003). Pair programming and pair trading: Effects on learning and motivation in a CS2 course. *Journal of Computing Sciences in Colleges, 18*(5), 49-56.

Demir, Ö., & Seferoğlu, S. S. (2017, October). *İşbirlikli problem çözmenin kodlama öğretimine yansıması olarak eşli kodlamanın incelenmesi*. Paper presented at the International Instructional Technologies &

Teacher Education Symposium (ITTES 2017), İzmir, Turkey. Abstract retrieved from https://www.researchgate.net/publication/321824838_Isbirlikli_Problem_Cozmenin_Kodlama_Ogretimine_Yansimasi_Olarak_Esli_Kodlamanin_Incelenmesi

Denner, J., Werner, L., & Ortiz, E. (2012). Computer games created by middle school girls: Can they be used to measure understanding of computer science concepts? *Computers & Education, 58*(1), 240-249. doi:10.1016/j.compedu.2011.08.006

Dongo, T., Reed, A. H., & O'Hara, M. (2016). Exploring pair programming benefits for MIS majors. *Journal of Information Technology Education-Innovations in Practice, 15*, 223-239. doi:10.28945/3625

Fessakis, G., Gouli, E., & Mavroudi, E. (2013). Problem solving by 5-6 years old kindergarten children in a computer programming environment: A case study. *Computers & Education, 63*, 87-97. doi:10.1016/j.compedu.2012.11.016

Grover, S., & Pea, R. (2013). Computational thinking in K-12: A review of the state of the field. *Educational Researcher, 42*(1), 38-43. doi:10.3102/0013189x12463051

Isong, B., Moemi, T., Dladlu, N., Motlhabane, N., Ifeoma, O., & Gasela, N. (2016). Empirical confirmation of pair programming effectiveness in the teaching of computer programming. In H. R. Arabnia, L. Deligiannidis, & M. Yang (Eds.), *Proceedings of the International Conference on Computational Science and Computational Intelligence (CSCI)* (pp. 276-281). IEEE/ Conference Publishing Services (CPS). doi:10.1109/csci.2016.59

Kalelioğlu, F., & Gülbahar, Y. (2014). The effects of teaching programming via Scratch on problem solving skills: A discussion from learners' perspective. *Informatics in Education, 13*(1), 33-50.

Lewis, C. M. (2011). Is pair programming more effective than other forms of collaboration for young students? *Computer Science Education, 21*(2), 105-134.

Lincoln, Y. S., & Guba, E. G. (1985). *Naturalistic inquiry.* Newbury, CA: Sage.

Maloney, J., Resnick, M., Rusk, N., Silverman, B., & Eastmond, E. (2010). The Scratch programming language and environment. *ACM Transactions on Computing Education (TOCE), 10*(4), 16. doi:10.1145/1868358.1868363

Martin, M. O., & Mullis, I. V. S. E. (2012). *Methods and procedures in TIMSS and PIRLS 2011: The TIMSS 2011 students confident in mathematics scale, eighth grade.* Retrieved from https://timssandpirls.bc.edu/methods/pdf/T11_G8_M_Scales_SCM.pdf

McChesney, I. (2016). Three years of student pair programming: Action research insights and outcomes In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education (SIGCSE)* (pp. 84-89). New York, NY: ACM. doi:10.1145/2839509.2844565

McDowell, C., Werner, L., Bullock, H. E., & Fernald, J. (2006). Pair programming improves student retention, confidence, and program quality. *Communications of the ACM, 49*(8), 90-95. doi:10.1145/1145287.1145293

Mentz, E., van der Walt, J. L., & Goosen, L. (2008). The effect of incorporating cooperative learning principles in pair programming for student teachers. *Computer Science Education, 18*(4), 247-260. doi:10.1080/08993400802461396

Millî Eğitim Bakanlığı. (2018). *Bilişim teknolojileri ve yazılım dersi öğretim programı*. Ankara: Millî Eğitim Bakanlığı. Retrieved from http://mufredat.meb.gov.tr/ProgramDetay.aspx?PID=374

Nagappan, N., Williams, L., Ferzli, M., Wiebe, E., Yang, K., Miller, C., & Balik, S. (2003). Improving the CS1 experience with pair programming. *SIGCSE Bulletin, 35*, 359-362. doi:10.1145/792548.612006

Nančovska, I., Kaučič, B., & Rugelj, J. (2008). Pair programming as a modern method of teaching computer science. *International Journal of Emerging Technologies in Learning, 3*(S2), 45-49.

Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas*. New York: Basic Books. doi:10.1007/978-3-0348-5357-6

Salge, C. A. L., & Berente, N. (2016). Pair programming vs. solo programming: What do we know after 15 years of research? In T. X. Bui & R. H. Sprague, Jr (Eds.), *Proceedings of the Hawaii International Conference on System Sciences* (pp. 5398-5406). IEEE. doi:10.1109/HICSS.2016.667

Swamidurai, R., & Umphress, D. (2015). Inverted pair programming. In *IEEE SoutheastCon-Proceedings*. IEEE. doi:10.1109/SECON.2015.7133010

Umapathy, K., & Ritzhaupt, A. D. (2017). A meta-analysis of pair-programming in computer programming courses: Implications for educational practice. *ACM Transactions on Computing Education, 17*(4), Article 16. doi:10.1145/2996201

Werner, L., & Denning, J. (2009). Pair programming in middle school: What does it look like? *Journal of Research on Technology in Education, 42*(1), 29-49. doi:10.1080/15391523.2009.10782540

Wiebe, E., Williams, L., Yang, K., & Miller, C. (2003). *Computer science attitude survey* (Report No. TR-2003-01). Raleigh, NC: NC State University.

Williams, L., & Upchurch, R. L. (2001). In support of student pair-programming. *SIGCSE Bulletin, 33*(1), 327-331. doi:10.1145/364447.364614

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. doi:10.1076/csed.12.3.197.8618

Wilson, A., & Moffat, D. C. (2010). Evaluating Scratch to introduce younger schoolchildren to programming. In J. Lawrance & R. Bellamy (Eds.), *Proceedings of the 22nd annual workshop of the psychology of programming interest group – PPIG2010*, 64-74. Retrieved from http://scratched.media.mit.edu/sites/default/files/wilson-moffat-ppig2010-final.pdf

Yıldırım, A., & Şimşek, H. (2016). *Sosyal bilimlerde nitel araştırma yöntemleri* (10th ed.). Ankara, Turkey: Seçkin Yayıncılık.

Yin, R. K. (2003). *Case study research, design and methods* (3rd ed.). Newbury Park: Sage Publications.

Yünkül, E., Durak, G., Çankaya, S., & Mısırlı, Z. A. (2017). Scratch yazılımının öğrencilerin bilgisayarca düşünme becerilerine etkisi. *Necatibey Eğitim Fakültesi Elektronik Fen ve Matematik Eğitimi Dergisi, 11*(2), 502-517.

Zhong, B., Wang, Q., Chen, J., & Li, Y. (2017). Investigating the period of switching roles in pair programming in a primary school. *Educational Technology & Society, 20*(3), 220-233.