ANADOLU UNIVERSITY | GRADUATE SCHOOL OF EDUCATIONAL SCIENCES
AJESI
ANADOLU JOURNAL OF EDUCATIONAL SCIENCES INTERNATIONAL

DOI: 10.18039/ajesi.1570205

The Impact of Solo and Pair Programming Modes on Problem-solving Skills and Motivation: Students' Reflections on Pair Programming

Ömer DEMİR[1]*, Murat ÇINAR[2], Süleyman Sadi SEFEROĞLU[3]

Date submitted: *22.10.2024* ***Date accepted:*** *17.07.2025* ***Type[4]:*** *Research Article*

Abstract

This study primarily investigates the effects of solo and pair programming on students' problem-solving skills and motivation. Additionally, it aims to identify key factors and tensions associated with pair programming based on students' reflections. An embedded mixed-methods research design was employed. The sample consisted of 42 undergraduate students. Participants in the control group, comprising 19 students, worked individually on computational tasks, while those in the experimental group engaged in the same tasks in pairs. All participants took part in computational thinking activities over seven weeks, structured around game development projects. Data were collected through questionnaires and in-depth interviews. The results indicated no statistically significant differences between the two groups in terms of overall motivation and problem-solving scores. However, the experimental group scored higher in the sub-dimension related to conceptualizing problems through modeling. Student motivation was found to be high across both groups following the intervention, regardless of the programming mode. After analyzing the qualitative data obtained from five students, a total of 122 codes were identified and categorized under six themes: 'team dynamics', 'task and platform characteristics', 'affective domain', 'cognitive strategies and problem-solving', 'pedagogical framework', and 'roles'. The qualitative findings suggest that the pedagogical benefits of pair programming can only be fully realized through the effective management of pairing strategies, peer communication, task difficulty, and collaborative processes.

Keywords: motivation, pair programming, problem-solving skill, solo programming, student reflections

Cite: Demir, Ö., Çınar, M., & Seferoğlu, S. S. (2025). The Impact of Solo and Pair Programming Modes on Problem-solving Skills and Motivation: Students' Reflections on Pair Programming. *Anadolu Journal of Educational Sciences International, 15*(3), 1145-1170. https://doi.org/10.18039/ajesi.1570205

[1]* Dr., The Department of Computer Technologies of Colemerik (Çölemerik) Vocational School of Higher Education, Hakkari University, Hakkari, Türkiye. omerdemir1986@gmail.com

[2] Dr., Borsa Istanbul Vocational and Technical Anatolian High School, Republic of Türkiye Ministry of National Education, Adana, Türkiye. murat_cinar@rocketmail.com

[3] Prof. Dr., The Department of Computer Education and Instructional Technology, Faculty of Education, Hacettepe University, Ankara, Türkiye. sadi@hacettepe.edu.tr

[4] This research study was conducted with Research Ethics Committee approval of Hacettepe University, dated 29.03.2018 and issue number 35853172/433-1408.

Introduction

Programming means to prepare a set of instructions that can be executed by a machine. Programming education is important in teaching students how to solve problems based on computational concepts and principles. In programming education, there is a growing interest in using collaborative programming as a pedagogical tool (Chorfi et al., 2020; Dybå Arisholm et al., 2007; Tan et al., 2024). Pair programming is one of the techniques used in collaborative endeavors in programming education. It involves two programmers working together on a single screen to fulfill a specific computational task (Bowman et al., 2020). In this process, one student plays the role of the driver, controlling the input devices, and conducting the programming processes (i.e., writing the codes), while the other undertakes the role of the navigator, guiding and supervising the driver by providing resources and offering guiding questions when necessary. Roles are interchangeable on a time or task basis. In a nutshell, each partner has responsibilities and tasks that need to be fulfilled on a rotating basis.

Numerous studies in the literature report positive student outcomes with regard to pair programming in terms of both cognitive and affective aspects (e.g., Çal & Gülcan, 2020; Demir & Seferoğlu, 2021a; Demir & Seferoglu, 2021b; Hawlitschek et al., 2023). Increasing course achievement, memory retention and code quality, reducing errors encountered in the programming process, experiencing high-level thinking processes, and increasing learner motivation, self-confidence, and satisfaction are just a few of them (Demir & Seferoğlu, 2021a; Hanks et al., 2011; Yang et al., 2016). Pair programming increases student interaction, provides opportunities for teamwork and collaborative learning, and facilitates the sharing of technical knowledge (Hanks et al., 2011; Xu & Correia, 2024). An important outcome is that students participate in gaining experience in collaboration that they need before entering the workforce (Demir & Seferoğlu, 2021b). In addition, students do not need to be in the same physical location to take advantage of pair programming, in that collaborative programming activities can be effectively conducted from distributed locations (Baheti et al., 2002; Lubarda et al., 2024). Hawlitschek et al. (2023) concluded from their review of the literature that pair programming produced more positive outcomes than solo programming in the case of university students. However, researchers also pointed to gaps in appropriate pedagogical frameworks and learning tasks. Balijepally et al. (2009) found that, regardless of task complexity, pair programming helps increase software quality and the programming confidence of low-performing students, and reduces the performance gap between them and high-performing students. Further studies also revealed that pair programming produces more effective outcomes in terms of computational thinking and programming, especially among less experienced students (Denner et al., 2014). This might be because novices (e.g., first-year students) are commonly more prone to failure due to inexperience in programming, a relatively low sense of belonging, and underdeveloped problem-solving strategies (Lubarda et al., 2024). In fact, the complex nature of programming leads to high dropout rates among first-year computing students (Navarro-Cota et al., 2025). This paves the way for the need for better guidance in computing endeavors.

Today, increasing digital data traffic, speed of technological development, variable characteristics of the target audience, complex business processes, and multidimensional analysis techniques have increased the demand for and dependence on software products. On the other hand, traditional software development practices have evolved into agile programming processes that are based on collaboration and self-organization among developers, and are more responsive to customer expectations and emerging technologies

(Balijepally et al., 2009; Mehta & Sood, 2023). Thus, collaborative programming enables the creation of large-scale or advanced products and shortens the development time.

The outputs of a programming education greatly depend on two fundamental factors: motivation and the ability to solve problems (Denner et al., 2021). However, there is also evidence in the literature to suggest that the impact of pair programming on problem-solving skills (e.g., Zhong & Li, 2020) and motivation (e.g., Krizsan & Lambic, 2024) remains limited. This is also valid for the general practice of programming that goes beyond the pair programming approach (e.g., Kalelioglu & Gülbahar, 2014; Psycharis & Kallia, 2017). The incongruence of the results found in the literature mainly stems from the social dynamics of pair programming affecting the collaboration process. The role of collaboration in programming education, therefore, needs to be further examined (Lai & Wong, 2020). In addition, the imbalance in workload distribution, irregularity in role switching between pairs, one-sided commitment, and low interaction between partners are among the main problems encountered (Tsompanoudi et al., 2015). The conflicting findings in the literature deserve attention as to how collaborative programming processes or practices are guided in terms of pair selection methods, team dynamics, programming knowledge and experience levels of partners, frequency of role change, workload distribution between partners, task difficulty, the pedagogical framework, etc. This is exactly why the social dynamics and the instructional framework are so important in pair programming. All this motivates us to explore whether pair programming provides a bonus effect in programming education. Although the current study examines the effectiveness of pair programming compared to solo programming in terms of problem-solving skills and motivation, it also explores the driving factors and tensions that influence the pair programming process.

Problem-Solving

Programming is inherently a problem-solving activity because programs are often written to solve problems. However, programming entails that the organization of ideas is grounded on computational principles. This necessarily involves the use of cognitive and affective tool sets that involve multi-level abstractions and representations that go far beyond knowing codes or combining them with a set of syntactic rules as we might when using a route map. With regard to this point, Wing (2006) pointed out that the approach conceptualized as computational thinking can be functional in terms of solving problems in different disciplines and also offers a universal attitude and skill set from which everyone can benefit. On the other hand, some studies indicate that programming commonly has a limited effect when it comes to making a significant difference in students' problem-solving skills (e.g., Çınar & Tüzün, 2021; Çiftci & Bildiren, 2020; Kalelioglu & Gülbahar, 2014). A similar trend applies to pair or solo programming modes. Zhong and Li (2020) found that students who worked in pairs achieved higher troubleshooting scores when making robotic artefacts, but their overall troubleshooting performance remained similar to that of solo programmers. This suggests that in some, but not overall, aspects, the pair mode in programming is a good catalyst for improving students' problem-solving skills.

Problem-solving approaches are not a single monolithic concept but involve different sets of cognitive and affective skills. Thus, it seems reasonable to examine sub-dimensions rather than meta-concepts related to problem-solving. In addition, despite the diversity of programming tools, environments, techniques, or activity types, the vast majority of studies in

the literature are based on solo programming activities. One of the emerging benefits of pair programming is that it encourages the verbal expression of mental processes during the problem-solving process. This facilitates reasoning, logical thinking, recognizing and thinking about intellectual models, exchanging ideas, and thus knowledge awareness and transfer. Peer interaction, which paves the way for capturing different perspectives with regard to the solution of a particular problem, provides significant educational benefits, especially in introductory programming courses (Hanks et al., 2011). The meta-analysis by Lai and Wong (2022) reported that collaborative problem-solving in programming led to better cognitive and affective learning outcomes than did solo programming activities.

Motivation

Students' commitment to engage in and sustain computational thinking while seeking solutions to problems is an important determinant of programming. Programming courses are based on practical applications that require reasoning with regard to abstract and complex concepts (Pilkington, 2018). Additionally, command expressions and the syntactic rules of programming languages pose an additional challenge. Programming courses are generally perceived by students as difficult and boring, resulting in low motivation and course completion rates (Alammary, 2019; Alturki, 2016; Noone & Mooney, 2018; Tsai, 2019). Pair programming makes the programming process more enjoyable through social interaction and peer collaboration (Hanks et al., 2011). However, excessive differences in the level of programming knowledge and poor communication within a student pair can lead to discomfort, which can negatively affect their effectiveness and motivation (Zarb & Hughes, 2015). Krizsan and Lambic (2024) reported no significant difference in motivation between solo and pair programming groups where pairs were matched with different matching patterns according to their skill level, including good-good, good-weaker and weaker-the lowest pairs. The topic of student motivation in the context of pair programming has received scant attention in the extant literature. Yang et al. (2016) reported that pair programming did not lead to a significant increase in student motivation in the ARCS model, except in terms of the confidence sub-dimension.

Although one-laptop-per-child projects have been gaining momentum around the world since the mid-2000s, most classroom arrangements require a few students to share a single device. While this is often seen as a factor that limits access to and use of digital resources in the classroom, it can create new pedagogical opportunities (Demir & Seferoğlu, 2017). At this point, traditional classrooms might lay the groundwork for collaborative pedagogies, which are often difficult to implement in education and allow for the structuring of knowledge through social interaction. In pair programming, only one team member uses the programming environment and hardware resources at a given time. In this regard, compared to the conventional classroom setting, the need for resources decreases and thus more students are included in the programming process. Moreover, these resources are not limited to screen-based programming tools but also apply to various physical programming hardware (e.g., robotic kits), especially those that are difficult to procure in the classroom. On the other hand, the joint engagement of the partners with computational problems as part of a collaborative process is an important parameter that determines the learning outcomes. Problems with task sharing, low task difficulty, social pressure on those who know less, time pressure, interruptions in the collaborative process, etc., can cause peers to drop out of the problem-solving process and miss out on the educational benefits of pair programming activities (Plonka

et al., 2012). Therefore, it is necessary to identify the experiences of partners in the peer programming process and the key factors that influence collaboration. In this context, the present study mainly addressed problem-solving skills and student motivation, which have contradictive findings in terms of their outcomes in programming modality, with their sub-dimensions. It also examines students' attempts to solve computational problems through pair programming and their reflections on the computational process in detail, such as the collaboration process, instructional setting, task type, etc.

Research Questions

The main aim of the present study was to scrutinize the effects of solo and pair programming modalities on motivation and problem-solving skills. It also aims to consider student reflection on pair programming as well as the key factors and tensions that influence collaboration. Based on these objectives, the following research questions were posed.

1. Is there a statistically significant difference between the motivation of the solo and pair programming groups?
2. Do the problem-solving skills of the solo programming group show a statistically significant improvement?
3. Is there a statistically significant improvement in the problem-solving skills of the pair programming group?
4. Are there any statistically significant differences between the problem-solving skills gained after solo and pair programming group activities?
5. What are the prominent pedagogical design elements in pair programming?
6. What are the major drawbacks and tensions that influence collaboration in pair programming?

Method

An embedded mixed research design (Creswell & Plano-Clark, 2006) was used in the study. This research design is preferred when more than one type of data needs to be collected due to the nature of the study. In the study, qualitative data were collected to facilitate the interpretation of the quantitative results. A quasi-experimental research design with a pretest-posttest control group was adopted in the quantitative dimension of the study. This experimental design involves non-random assignment of participants to the test groups (Fraenkel et al., 2012). Participants were not randomly assigned to the experimental group in the study.

Study Group

A total of 42 volunteer senior students enrolled in an elective course at the Computer Education and Instructional Technology (CEIT) division of a public university in Türkiye participated in the study. Of these, 19 were placed in the control group (4 female, 15 male), and 23 were in the experimental group (10 female, 13 male). To compare the groups in terms of their programming self-efficacy, the Mann–Whitney U test was employed. As a result of the Mann–Whitney U test, no statistically significant difference was found between the control group (mean rank = 21.47) and the experimental group (mean rank = 21.52) in terms of programming self-efficacy (Z = .013, p = .990). Similarly, the comparison of problem-solving

skills also revealed no statistically significant difference between the control group (mean rank = 21.34) and the experimental group (mean rank = 21.63) (Z = .076, p = .940). For the qualitative part of the study, five students (three female and two male) from the experimental group were selected based on gender and pre-test scores, using the maximum diversity method.

Data Collection Tools

Five data-collection instruments were used in this study. These tools were 'personal information form', 'programming self-efficacy scale', 'motivation scale', 'problem-solving skill scale', and lastly 'semi-structured interview form'.

Personal Information Form

This form includes questions about the participants' gender, their block-based programming experience, game development experience, their number of programming-related courses they have taken, etc.

Programming Self-Efficacy Scale

The original form of the programming self-efficacy scale was developed by Ramalingam and Wiedenbeck (1998). This consisted of 32 items under four factors. This scale was adapted to Turkish by Mazman (2013). The new form of the scale consisted of nine items with two factors. These factors were *performing simple programming tasks* and *performing complex programming tasks*. The overall Cronbach Alpha reliability coefficient of the 7-point Likert-type scale was calculated as 0.93 in both the original and this study.

Motivation Scale

The scale, originally developed by Keller (1987), consisted of 36 items under four factors. The scale was developed on the theoretical basis of Keller's ARCS motivation model. Kutu and Sözbilir (2011) reported that the Turkish version of the scale consisted of 24 items under two factors. These factors were *attention-relevance* (11 items) and *confidence-satisfaction* (13 items). The overall reliability coefficient of the Turkish form of the original scale was calculated as 0.83, while it was 0.92 in this study. The scale was designed as a 5-point Likert type (1 = I totally disagree, 5 = I totally agree). Consequently, it is possible to obtain a maximum of 120 points, and a minimum of 24. Higher scores indicate higher motivation. 'I studied with pleasure' is a typical sample item of the scale.

Problem-Solving Skill Scale

The problem-solving skill scale was developed by Yaman and Dede (2008). The scale contains 18 items under five factors. These factors were 1) thinking about the effects of the solution to the problem (five items), 2) problem-solving through modeling (three items), 3) probing alternative solutions (four items), 4) commitment to apply the determined solution (three items), and 5) analyzing the problem encountered (three items). The scale explained

82% of the total variance. The overall Cronbach Alpha reliability coefficient of the original scale was reported as 0.88, whereas it was reported as 0.91 in this study. The scale was a 5-point Likert type (1 = Never, 5 = Always). Consequently, it is possible to obtain a maximum of 90 points, and a minimum of 18. Higher scores signify higher problem-solving skills. 'I consider every solution when solving a problem' is a sample item of the scale.

Semi-Structured Interview Form

The semi-structured interview form comprised eight questions. This form was revised after being checked by two subject-matter experts in the field of instructional technology. The wording of the interview questions was revised to improve clarity in line with expert opinions. The interview form includes a set of questions addressing pair selection and partner characteristics, cooperation, task sharing, problems experienced in pair programming, and finally the effect of pair programming on coding motivation, skill development, and code quality (See Appendix 1).

Experimental intervention

Learning Tasks

The Scratch 3.0 environment was preferred as the programming environment. Scratch was used because it is a block-based tool with a free, easy-to-use and rich visual toolkit. Students were given a different arcade game programming task each week. In both the experimental and control groups, the game development was started by the instructor in the first class. Subsequently, the students were asked to code the remaining part during the second lesson by conforming to the predetermined features of the game. In this process, the instructor was present to guide the students.

Implementation of Pair Programming

Before implementation, general introductory information about pair programming was given to the students in an effort to make it more fruitful. Subsequently, dyads were formed for pair programming in the experimental group. For this, each student was asked to choose different peers with whom they would like to work. The students were matched to one of these students each week. Those who had been matched once were not allowed to be re-matched. This was done because the students were expected to experience unique interactions with different peers. Programming dyads changed their driver and navigator roles every 20 minutes.

Data Collection Process

To begin with, ethical approval was obtained from the ethics committee of the university where the research was conducted. Prior to computational activities, the Scratch environment and its basic features were introduced to the students for a week as part of the programming course. Then, a personal information form, pre-test of the problem-solving skill scale, and the programming self-efficacy scale were administered to students. In addition, there was a random selection of which group would be the experimental group prior to implementation. Then, the seven-week implementation was performed. At the end of the implementation

process, students' problem-solving skills were re-measured in addition to their motivation. Finally, semi-structured interviews were conducted with the five students (three female and two male) from the experimental group. The experimental design of the research is shown in Table 1. One of the researchers was the faculty member teaching the course, while one of the other researchers assisted in the above-mentioned course and collected the research data.

Table 1
Details of the Experimental Research Design.

Groups	Pre-test	Experimental intervention	Post-test	Interview
Control	Personal information form, Problem-solving skill, Programming self-efficacy	Solo programming	Problem-solving skill, Motivation	-
Experiment	Personal information form, Problem-solving skill, Programming self-efficacy	Pair programming	Problem-solving skill, Motivation	Semi-structured

Data Analysis

In the quantitative analysis, MS Excel and IBM SPSS Statistics (ver. 24) were used. Prior to analysis, missing data was filled using the linear trend method. Then, factor and overall scale scores were obtained by taking the average of the relevant items. While the groups were considered as independent variables, motivation, and problem-solving skills were considered as dependent variables. Since the data set was small, non-parametric statistical analysis methods were used. Frequency, arithmetic mean, and standard deviation were used to describe the data. The Wilcoxon Signed Ranks Test was used to examine the development of each group. The gain scores were obtained by subtracting the pre-test from the post-test scores. The gain scores of the groups were compared with the help of the Mann-Whitney U test. The significance level was determined as .05. When a statistically significant difference was achieved, the r effect size was calculated to help interpret the significance in practice and was interpreted according to Cohen (1998). In the qualitative analysis, semi-structured interview recordings were transcribed by an experienced student for a fee. This transcript comprised 32 pages, 8,200 words and 42,911 characters (without spaces) with the format of 12 point, Times New Roman font and 1.5 line spacing. Qualitative content analysis was applied to the transcript. With regard to the content analysis, qualitative data were analyzed in depth without predetermined concepts in mind so that relationships existing in the data were unearthed (Strauss & Corbin, 1990). Two experts were involved in the qualitative analysis process. Both experts totally agreed on the codes after discussing the disputed codes in order to ensure inter-coder reliability. The NVivo (ver. 10) qualitative analysis tool was used in the process. Open, axial, and selective coding were used during the analysis process. After free coding, a total of 224 free codes were obtained. The codes were then revisited, with 55 codes deleted, and a total of 45 code pairs or trios were merged. At the end of this process, the remaining 122 codes were grouped into six themes.

Findings

The findings were reported in the order of the research questions.

The Difference Between the Motivations of the Solo and Pair Programming Groups

The results demonstrated that there was no statistically significant difference between the solo and pair programming groups in terms of attention-relevance (Z = 1.166, p = .244), confidence-satisfaction (Z = 1.292, p = .196) factors, and motivation in general (Z = 1.267, p = .205) (See Table 2).

Table 2
Results of the Mann-Whitney U Test on the Motivation Differences of the Solo and Pair Programming Groups

Motivation	Groups[1]	M[2]	MR[3]	SD	Z	p
Overall	Control	3.79	18.87	.67	1.267	.205
	Experiment	4.08	23.67	.48		
1) Attention-relevance	Control	3.73	19.08	.72	1.166	.244
	Experiment	4.04	23.50	.61		
2) Confidence-Satisfaction	Control	3.85	18.82	.72	1.292	.196
	Experiment	4.28	23.72	.50		

[1] N_{ctrl} = 19, N_{exp} = 23, [2] The scale is 5 Likert type, [3] MR = Mean Rank.

The Gain in the Problem-Solving Skills of the Solo Programming Group

The solo programming experience did not cause a statistically significant gain in thinking about the effects of the solution of the problem (Z = 1.117, p = .264), problem-solving through modeling (Z = .175, p = .861), probing alternative solutions (Z = .810, p = .418), commitment in applying the determined solution (Z = .860, p = .390) factors and overall problem-solving skill (Z = 1.025, p = .305). When it comes to the factor of analyzing the problem encountered, a statistically significant gain of medium effect size was found (Z = 1.960, p = .050, r = .318) (See Table 3).

Table 3
Results of the Wilcoxon Signed-Rank Test on the Gain in the Problem-Solving Skill of the Solo Programming Group

Problem-Solving Skill	Groups	M[1]	SD[1]	Ranks[2]	Z	p	r
Overall	Pre-test	4.02	.52	NR=6 PR=12 T=1	1.025	.305	N/A
	Post-test	4.20	.43				
1) Thinking about the effects of the solution of the problem	Pre-test	4.02	.57	NR=6 PR=11 T=2	1.117	.264	N/A
	Post-test	4.24	.64				
2) Problem-solving through modeling	Pre-test	3.98	.68	NR=8 PR=7 T=4	.175	.861	N/A
	Post-test	3.95	.52				
3) Probing alternative solutions	Pre-test	3.76	.76	NR=8 PR=9 T=2	.810	.418	N/A
	Post-test	3.99	.65				

4) Commitment to apply the determined solution	Pre-test	4.30	.60	NR=7 PR=7 T=5	.860	.390	N/A
	Post-test	4.49	.50				
5) Analyzing the problem encountered	Pre-test	4.12	.51	NR=4 PR=12 T=3	1.960	.050*	.318
	Post-test	4.39	.46				

** Significant at the level of .05, Nctrl = 19, [1] The scale is 5 Likert type. [2] NR = Negative Ranks (Pre-test > Post-test), PR = Positive Ranks (Post-test > Pre-Test), T = Ties (Post-test = Pre-test).*

Gain on the Problem-Solving Skill of the Pair Programming Group

A statistically significant problem-solving skill gain in the case of the pair programming group was achieved in terms of thinking about the effects of the solution of the problem (Z = 2.625, p = .009, r = .387, medium effect size), problem-solving through modeling (Z = 3.680, p = .000, r = .543, large effect size) factors and problem-solving skill in general (Z = 2.311, p = .021, r = .341, medium effect size). No statistically significant finding was obtained in terms of probing alternative solutions (Z = 1.858, p = .063), commitment to apply the determined solution (Z = .000, p = 1.000), and analyzing the problem encountered (Z = .318, p = .751) factors (See Table 4).

Table 4
Results of Wilcoxon Signed-Rank Test on the Gain of Problem-Solving Skill of the Pair-Programming Group

Problem-Solving Skill	Groups	M	SD	Ranks	Z	p	r
Overall	Pre-test	4.04	.52	NR=9 PR=13 T=1	2.311	.021*	.341
	Post-test	4.33	.43				
1) Thinking about the effects of the solution of the problem	Pre-test	3.97	.67	NR=4 PR=16 T=3	2.625	.009**	.387
	Post-test	4.40	.52				
2) Problem-solving through modeling	Pre-test	3.72	.63	NR=1 PR=18 T=4	3.680	.000**	.543
	Post-test	4.28	.43				
3) Probing alternative solutions	Pre-test	3.76	.69	NR=6 PR=12 T=5	1.858	.063	N/A
	Post-test	4.11	.67				
4) Commitment to apply the determined solution	Pre-test	4.51	.57	NR=8 PR=7 T=8	.000	1.000	N/A
	Post-test	4.51	.50				
5) Analyzing the problem encountered	Pre-test	4.38	.55	NR=7 PR=7 T=9	.318	.751	N/A
	Post-test	4.41	.56				

** Significant at the level of .05, ** Significant at the level of .01, Nexp = 23.*

The Difference Between the Problem-Solving Skill Gain of the Solo and Pair Programming Groups

In terms of problem-solving skills in general (Z = .443, p = .658), thinking about the effects of the solution of the problem (Z = .903, p = .367), probing alternative solutions (Z = .584, p = .559), commitment to apply the determined solution (Z = .542, p = .588), and analyzing the problem encountered (Z = 1.432, p = .152) factors, no statistically significant

difference was noted between the problem-solving skill gain of the solo and pair programming group (See Table 5). In terms of the problem-solving through modeling factor, a statistically significant difference of medium effect size was found in favor of the experimental group (Z = 3.005, p = .003, r = .464).

Table 5
Results of the Mann-Whitney U Test on the Differences in Problem-Solving Skill Gain of the Solo and Pair Programming Groups

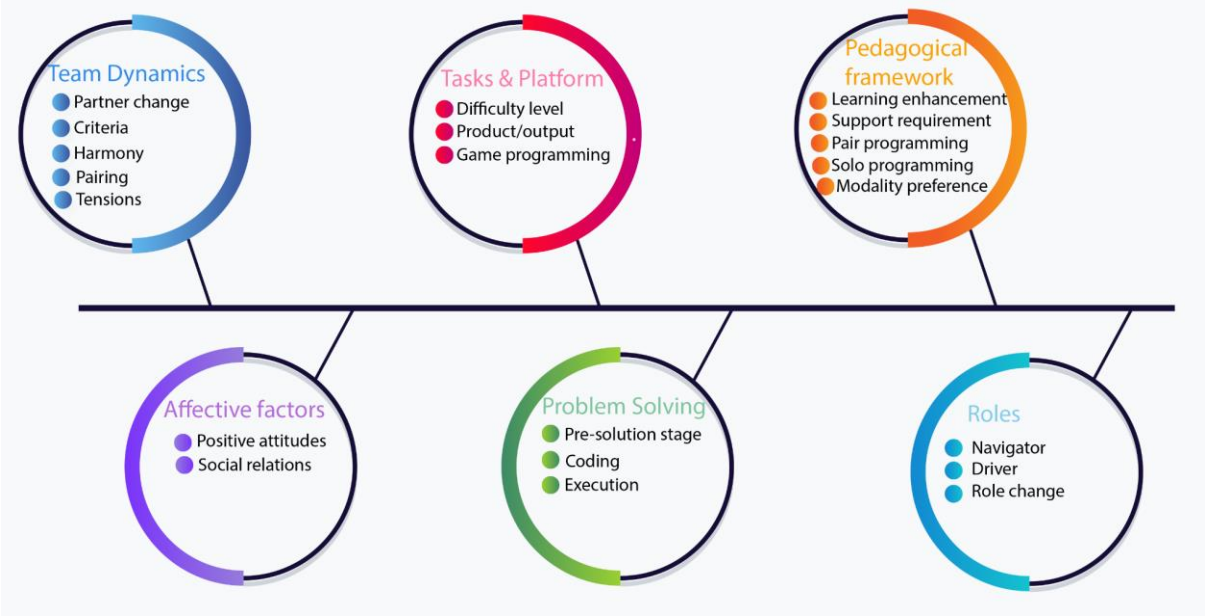
Problem-Solving Skill	Groups	Gain score[a]	MR	SD	Z	p	r
Overall	Control	.18	20.58	.54	.443	.658	N/A
	Experimental	.29	22.26	.53			
1) Thinking about the effects of the solution of the problem	Control	.22	19.63	.75	.903	.367	N/A
	Experiment	.43	23.04	.69			
2) Problem-solving through modeling	Control	-.04	15.37	.42	3.005	.003**	.464
	Experiment	.41	26.57	.44			
3) Probing alternative solutions	Control	.22	20.29	.91	.584	.559	N/A
	Experiment	.35	22.50	.80			
4) Commitment to apply the determined solution	Control	.19	22.61	.76	.542	.588	N/A
	Experiment	.00	20.59	.67			
5) Analyzing the problem encountered	Control	.26	24.42	.52	1.432	.152	N/A
	Experiment	.03	19.09	.60			

** *Significant at the level of .01.* [a] *Post-test - pre-test, Nctrl = 19, Nexp = 23.*

Students' Reflections on and Prominent Pedagogical Issues Related to Pair Programming

Students' perceptions of the motivational and problem-solving effects of pair programming, and the prominent pedagogical issues that emerged during pair programming, are the other research questions of the study. To answer these questions, the interview data were analyzed using qualitative content analysis involving a ground-up approach. The 122 free codes revealed after qualitative data analysis were grouped into 21 categories under six themes (See Figure 1). These themes could be listed as team dynamics (32 codes), task and platform characteristics (10 codes), affective domain (10 codes), cognitive strategies and problem-solving (19 codes), the pedagogical framework (40 codes), and roles (11 codes).

Figure 1
Themes and Categories Emerging from the Qualitative Analysis



With regard to the theme of team dynamics, there are five categories: partner change, criteria, team harmony, pairing, and tensions. Although changing partners was not welcomed at the beginning of the implementation, it was later perceived as beneficial for enhancing peer learning and minimizing persistent interpersonal issues within teams. However, it should be noted that pair change frequency (such as at the task level, or on a weekly or monthly basis) is critical. At this point, it would be more reasonable to change pairs every couple of weeks instead of after each session or task. The partner's interest in the topic, and being thoughtful and open to learning are the most prominent criteria for choosing a partner. The factors affecting team harmony are knowledge-level differences between peers, task sharing, peer communication, and rules. It was observed that peer-to-peer communication creates a synergistic driving force in terms of task performance. The task share and accompanying responsibilities here generally were seen not to change throughout the problem. However, the students frequently reflected on the exchange of ideas during the tasks. One of the partners was more related to the learning environment (e.g., subject matter, teacher, peers), and the other was more focused on the problem itself. The difference in the level of subject matter knowledge between the team members came to the fore as a factor that positively contributed to team harmony and task engagement or lack thereof. However, this difference should remain at such a level that it does not cause any of the team members to become disengaged from the task, and allows the members to learn from each other. The rules among the partners are very functional in terms of preventing tensions. Peer communication ensures a clear understanding of the messages to be conveyed and increases cooperation within the pair. It was observed that when communication is poor, tasks are carried out with individual effort and sometimes left unfinished.

> *P1: Again, like I said, I couldn't tell him [his partner] to get on with it and leave the other thing, because I didn't have a lot of communication and I wasn't very close to him. He kept doing his job, I kept writing code on Scratch. Something like that happened that week, I couldn't finish the issue [task] that week, it was half done.*

Low intimacy between team members led to the need for increased explanation, the personalization of criticism, and a reluctance to communicate. Grade level is very critical for pair members to get to know each other. Especially in the lower classes, the time needed for the pair to get to know each other can increase. Students found it beneficial to partner with their best friends and to work with their partners for longer periods of time. One student found that being paired with more knowledgeable others led to increased effort and better learning. Students also addressed that there are also risks in terms of good communication with the partners they are forced to be paired with. At this point, it is recommended that the student be given the choice of partner as much as possible. The main causes of tension were communication problems, the perception of insufficient effort on the part of the partner, and the overburdening of one of the partners during the task.

P3: Because I was in a team with some people who knew more [high prior knowledge]. In this way, I made more effort to compensate for my own deficiency.

Students reported that initially the task difficulty was at a level they could manage, but became cognitively challenging in the later stages. Lack of peer support at the desired level led to difficulty in completing tasks in some cases, increased the time spent on the task, and even resulted in disengagement from the task. The fact that the learning tasks are artifact-based allows students to reflect on their knowledge and makes it more visible. Pair programming has been reported to affect the quality of artifacts as well as task performance and efficiency. On the other hand, the fact that the tasks are artifact-based has raised questions about how to understand the nature and quality of the work (e.g., code length). The students stated here that the complexity of the problem and its openness to more than one solution highlighted the quality of the computational solutions. Game design and programming activities were found to be very effective in this regard, providing students with opportunities for authentic work. At this point, Scratch and the ready-made object library made it easier to implement computational strategies and to create artefacts that were more aesthetically pleasing and appropriate. Interestingly, students indicated that pair programming was more suitable for in-class computational tasks, such as problems and projects, and content learning, but not for out-of-class tasks, including homework, projects, and presentations.

P4: That's why I think the purpose is important here. If we do something to learn, I prefer to do it with a partner. But if I have to present something, for example, I would prefer to do this project assignment and final project individually. However, personally, I would prefer to work with my peers during class.

Students' negative attitudes and prejudices with regard to pair programming prior to the intervention changed over time. Students reported that pair programming made the process of solving computational problems more enjoyable when the pair was in harmony, which had a positive impact on their motivation and also increased their belief in task performance. Pair programming increased the sense of community in the class. However, those in the navigator role within the group sometimes felt left out.

P5: Well, at first I thought pair programming would be bad... But later, when I came to the class and found a partner, I saw that it [pair programming] is not like that.
P3: It was more fun to chat and code [with someone] at the same time. If I was alone, I might get bored and break away [from the task] after a while.

In the pre-solution phase of computational problems, analyzing the problem, envisioning the solution, and creating algorithms became prominent cognitive processes. In

pair programming, sharing ideas with a partner, such as saying them out loud, makes the students aware of their thoughts about the task or problem and allows them to think about them. It also reduced the trial-and-error approach to finding solutions and allowed more time for thinking (analysis, decomposition, creating algorithms, etc.). Students emphasized that pair programming increases the diversity of ideas at every stage of problem-solving. However, one participant noted that the diversity of ideas about the problem also caused complexity, slowed the transition to the solution-stage, and required going back to the beginning (understanding the problem phase).

> *P4: I can say that I learn more when I program with a partner. Because in one of these modalities, namely if I were alone, I would look at the problem and try to find a solution. If I could not, I would look again and try to find a solution and think on my own. But in pair programming, there were times when I would say it out loud to someone, I would hear it from someone else, and I would reinforce my learning by explaining it to someone else.*
> *P5: First, we discussed how we could do it, but I didn't do it directly like, let's take this [code block] and put it just here.*

Students found pair programming beneficial in terms of code quality and efficiency, especially in using less code. The selection of appropriate code blocks, the removal of unnecessary codes, and the shortening of codes stood out as factors that increased efficiency. It also contributed to the rapid implementation of the developed strategies into appropriate code blocks. In solo programming, students generally applied the first solution that came to mind; however, in pair programming, they spent more time thinking while sharing strategies with their partners, providing an opportunity to revise ideas.

> *P4: If I were alone [solo], I would look at [the problem] and try to find a solution. If I couldn't, I would look again and try to find a solution [trial and error] and think on my own. But in pair programming, there were times when I would say it out loud, and there were times when I would hear it from someone else, and I would reinforce my learning by explaining it to someone else.*

Contrary to usual computational problems, game programming in pairs allows students to address different user behaviors, and design issues and thus increase the quality of the codes by adding new functionalities. Here, game programming has also paved the way for creative work to emerge as part of the design process. However, the increase in quality has been limited in computational problems where there is no design element, and where there is only one correct answer. In addition, the pair programming mode offers a variety of ideas when it comes to designing the user interface, such as placing design objects, beyond just coding. However, although this mode might increase the task completion rate and catalyze the search for a solution, the diversity of ideas sometimes leads to additional workload and loss of time. In addition, there has also been criticism that as the number of code blocks increases, pair programming creates more distractions and reduces the control of code blocks.

> *P2: As two coders, we come up with an idea and build the code blocks and a game. Well, this also enables higher quality.*

Regarding the pedagogical framework, the prominent issues are learning enhancement, support requirements, pair and solo programming, and programming modality preference. Pair programming encourages peer learning, the scaffolding of subject learning, the opportunity to acquire a better understanding, self-evaluation, and the closing of subject

gaps. In addition, the vocal expression of thoughts helps students to retain, remember, and reinforce what has been learned. However, the lack of opportunities for practice in the navigator role was mentioned as a problem. There are also negative impressions. Students have stated that pair programming can increase teacher workload and cause some difficulties in terms of classroom management. In addition, pair programming can be a bit time-consuming in terms of problem-solving.

> *P3: The classmate next to me who knew better [pair programming] told me the key issues, what they were for, and where to use them. It is difficult for an instructor to take care of others [the whole class] and deal with everyone personally.*
> *P5: The topics stayed better in my mind when my colleague told me things I didn't know or had missed.*

Students' support needs became more evident as they progressed through the learning topics. Students had asked for and expected support, especially from their teammates, other classmates, and the instructor. They reported that when they had difficulty solving computational problems, they sought external resources and received support from other teams. Team mates stated that they complemented each other, and corrected each other's deficiencies, especially during the programming process. At the same time, the knowledge gap between teammates decreased over time. The shared decision-making in pair programming is reported to increase the quality of the resulting product; however, conflicting views have emerged about the number of computational problems solved per unit of time. Team building with diverse individuals in pair programming improves their social and teamwork skills. In addition, it has been noted that pair programming is reported to increase individual effort. However, prior knowledge stood out as a determining factor in this increased effort. Students emphasized that this increase in effort would not be the case for a student with insufficient knowledge of the material. Pair programming is said to improve programming skills and task performance, while sharing ideas and building common understanding encourages them to tackle more complex problems. One of the students recommended shaping coding modality according to purpose. Some students, especially those who used to learn individually, stated that they prefer pair programming for computational tasks, especially for learning purposes, but solo programming for student assessments such as homework/projects. This also shows that students' test anxiety affects their modality preferences.

> *P4: You know, we solved it together with a person I am not very close to... If two people agree on something without knowing anything, they can handle these [problems].*

In pair programming, students participate in teamwork in the roles of navigator or driver. The students in the navigator role played a bridging role in the team's communication with the other teams and the instructor. When teammates were engaged in developing strategies for computational problems, students in the driver role were more interested in translating strategies into code. Students stated that if their partner in the navigator role did not have sufficient prior knowledge, they would remain passive and might have difficulty learning the topic just by following the code. Students perceive that those in the driver role become more familiar with the code and learn better than their partner. In terms of role distribution, it was observed that the first student to sit in front of the computer preferred to be the driver. This means that the driver role is more popular and prioritized among students. Role change was typically done based on the course session with, for example, one role change per course hour. However, when role switching was based on the class session, it was found that it was

difficult for the new driver to analyze the existing codes and continue based on those codes. In this case, a more reasonable approach was to change roles based on tasks rather than time.

> *P2: I feel like I am learning when I control the keyboard and mouse. On the other hand, when my pair buddy controls the devices, I feel like he is managing the process and only he is learning.*
> *P1: You know, when my partner sits at the computer in the second lesson, it might be a little difficult for him to understand the code blocks.*

Discussion

Pair programming is a form of collaborative learning. Students in pairs learn from each other. Peer tutoring that is also seen in pair programming is one of the most effective forms of collaborative learning based on empirical results (Yang et al., 2016). However, uncovering the pedagogical dynamics underlying pair programming will also reveal how instructional interventions should be designed.

The Effect of Pair Programming on Motivation (RQ1)

Motivation is an internal process stimulating an individual to meet their needs (Lussier, 1996). It is one of the cognitive structures that acts as a driving force for effective learning. However, there are very few quantitative studies on the impact of pair programming on motivation. In the current study, there was no significant difference between the motivation scores of the test groups (RQ1). However, an examination of the results reveals that the motivation scores of the experimental group were relatively higher than those of the control group. The average item score above four on a 5-point scale also indicates that the learner's motivation after the computational activities is quite high in the case of the experimental group. Similarly, the average score of the comparison group was above three (M = 3.79>3.00), which indicates that the motivation of the students in the control group was also high. From these results, it can be concluded that the motivation of the students with regard to the computational activities is high, regardless of the modality. In addition, game-based design and programming incorporate highly motivating elements (Demirkıran & Hocanin, 2021) such as rich multimedia objects, concrete elements, rules, stages, score tables, rewards, storyboards, and learning scenarios to foster computational thinking skills. These gameplay elements may have influenced the students' motivation scores in our study.

Yang et al. (2016) examined the effects of pair programming activities on motivation using the ARCS model. The results showed that participants in the pair programming group had higher levels of (self) confidence than those in the solo programming group. On the other hand, there were no significant differences between the groups in terms of the other three dimensions in the form of attention, relevance, and satisfaction. On the other hand, when evaluating the results regarding the pedagogical outcomes of pair programming, the way dyads are formed stands out as an issue that needs to be considered. Unlike the voluntary pairing in our study, the pairing method in Yang et al.'s study was based on programming skills. Thus, students with higher programming scores were asked to pair up with those with lower programming scores. The positive impact of pair programming on students' confidence has been over-emphasized so far. For instance, another study reported that students in the pair programming group enjoyed computational activities more and had greater confidence than

those in the solo programming group (Bishop-Clark et al., 2006). Krizsan and Lambic (2024) also reported that solo and pair programming modalities did not cause any significant difference in student motivation levels, although they did observe an increase in efficiency in terms of pair programming. The overall results overlap to some extent with the results of our study.

Investigation of Students' Problem-solving Skills According to Solo and Pair Programming Modalities (RQ2, RQ3, and RQ4)

Pair programming requires students to get together to construct co-meanings and engage in computational thinking and problem-solving collaboratively (Xu et al., 2023). Problem-solving skill involves knowing what to do in uncharted territory (Jonassen, 2000). One of the crucial functions of education is to help students develop the attitudes and strategies they will use to deal with problems they will face in the later stages of their lives. Programming is essentially a problem-solving process that involves reframing a problem in a way that a computer can interpret and execute (Fanchamps et al., 2021). The mindset used in programming can be applied to a wide range of problems encountered in other areas of life or daily routines. In line with this idea, the current study also evaluated the impact of computational thinking and programming on problem-solving skills. Well-structured problems (i.e., problems in which the steps to be taken and the overall result are well-defined) are not sufficient to prepare students for problems in real and professional lives. In contrast to concept or rule learning, programming instruction includes the use of various analyses based on an abstraction of multiple inputs and hierarchical task decomposition to develop problem-solving skills. Furthermore, design problems are inherently complex and ill-structured because there are no clear goals and no limited and definitive solution, and they often require the integration of more than one discipline (Jonassen, 2000). Design thinking is a human activity that paves the way for creative ideas in generating solutions to problems (Razzouk & Shute, 2012). In this study, students engaged in computational thinking based on game design activities. In the control group, there was no significant change observed between the students' problem-solving overall and sub-factor scores (except for analyzing problems) following the computational activities required for game programming (RQ2). However, students' analysis competence regarding problems was significantly enhanced in the control group. On the other hand, students' problem-solving competencies improved significantly after the computational activities in the pair programming group (RQ3). It is noteworthy that the factor scores related to modeling the problem, conceptualizing it in some way, and understanding the implications of problem-solving increased significantly in terms of the subfactors in the experimental group. When it comes to group comparisons, there is no statistically significant difference between the scores of the solo and pair programming groups with regard to overall problem-solving scores and the subfactors, with one exception (RQ4). This exception is the modeling of the problem, which was higher in the case of the pair programming group.

Our results have both overlapping and differing aspects with regard to the relevant literature. In essence, collaborative engagement with computational problems can be beneficial for students when it comes to acquiring cognitive, affective, and interpersonal skills. A meta-analysis has revealed that collaborative problem-solving in computer programming is more effective than computational thinking related to individual problem-solving in terms of cognitive (decomposition of the problems, design of the solutions, creating artifacts, abstraction, troubleshooting, generalization, etc.), and affective (engagement, positive

feelings, self-efficacy, satisfaction, interest, etc.) learning outcomes (Lai & Wong, 2022). However, a similar effect was not found for the social aspect. In addition, task design, programming environment, and duration have attracted attention as having a moderating effect on the attainment of these outcomes. Kazimoglu et al. (2012) reported that students found serious games beneficial in terms of improving their problem-solving skills in the context of an introductory programming course at the undergraduate level. There are also contradictory findings on the effects of programming activity on problem-solving competency in the literature. Kalelioglu and Gülbahar (2014) reported that block-based programming activities did not make a significant difference to the problem-solving skills of 5th-grade students. That study was based on solo programming activities in primary education. Another study found that while engagement in programming activities improved students' problem-solving scores in math, it did not result in a significant difference compared to those who did not participate (Psycharis & Kallia, 2017). Previous reports have also disputed the benefits of pair programming over solo programming in terms of the cognitive aspects of learning. For instance, Harsley et al. (2017) reported no significant differences in learning gains between pair and solo programmers using an intelligent tutoring system. However, the students working in pairs completed the task faster and more efficiently. Similarly, Demir and Seferoğlu (2021a) found no difference between the two modalities regarding code quality and achievement.

Pedagogical Design (RQ5), and Major Drawbacks and Tensions in Pair Programming (RQ6)

The discrepancies between the results raise the question of how to appropriately incorporate pedagogies into programming instruction. Although comparative studies suggest that pair programming produces more positive outcomes than solo programming, the appropriate instructional design and task types are questions that remain to be answered (Hawlitschek et al., 2023). Therefore, studies in the literature on the effects of pair programming on motivation and problem-solving should be approached with caution. This is because the instructional design settings could be argued to be the most important factor in determining the pedagogical benefits of pair programming.

In the current study, based on the qualitative findings of the pair programming process based on student reflections, the prominent pedagogical design elements identified were as follows (RQ5): (1) team dynamics, (2) task and platform characteristics, (3) affective domain, (4) cognitive strategies and problem-solving, (5) pedagogical framework, and (6) roles (See Figure 1). Therefore, these issues need to be addressed while considering the previous findings. A literature review that examined 61 experimental studies of pair programming between 2010 and 2020 underlines a need for research on the knowledge, guidelines, and problem-solving strategies used during tensions that occurred in teams to obtain a deeper understanding of effective instructional designs in this programming modality (Hawlitschek et al., 2023).

The qualitative part of this research sheds light on the authentic experiences of students during the pair programming. The main outcome of tracing this authentic experience was the identification of major drawbacks and tensions that emerged during the process (RQ6). The tensions were found to be mainly due to prejudice against pair programming, low intimacy, and communication problems between the partners, pair incompatibility, imbalance in task effort, and disengagement from the task. Although there was some initial reluctance,

prejudices against pair programming diminished over time in the study. In accordance with the qualitative results, the most significant indicator of team harmony was identified as dialogue between pairs. The lack of dialogue between pair members during pair programming constitutes a challenging scenario. The prevailing academic consensus identifies low intimacy and communication as a significant barrier to the pedagogical outcomes of pair programming and as one of the main contributors to failure (Zarb & Hughes, 2015). In addition, dialogue patterns also affected the quality of pair programming (Tan et al., 2024).

The biggest obstacles to students working in pairs are achieving team harmony and scheduling the programming process (Hanks et al., 2011). We concluded that pairing strategy, role changes, and student characteristics are all determinants of team compatibility. The frequency of partner change, communication between pair members, differences in knowledge levels, distribution of responsibilities, and rules are also pillars of team harmony and tension management. In particular, rules and criteria served not only to mitigate tensions but also to improve communication. These results are consistent with the literature to a certain extent. Criteria and guidelines were reported to improve communication between partners (Zarb & Hughes, 2015). Pair incompatibility can cause problems in the classroom, which can lead to an increase in teacher workload. Student behavior, communication, and socio-emotional states in pair programming lead to the formation of different collaboration patterns, which in turn affect the quality of collaboration (Xu et al., 2023). Xu et al. (2023) reported that the pairs which exhibited consensus patterns and achieved group regulation showed the highest cooperation performance and outcomes. Team harmony and awareness also ensure that task-related cognitive load is distributed among team members (Zhong & Wang, 2021). When there was a lack of communication and mental effort within a pair, it was observed that tensions arose and there was a disengagement from the task. As the present study also found, Bowman et al. (2020) posit that the workload is not equitably distributed between the partners and that ethical differences among paired students can engender tensions and reduce the effectiveness of pair programming.

It is important to keep pairing as optional as possible to minimize communication problems. Students found it more beneficial to partner with their best friends. However, Demir and Seferoğlu (2021b) revealed that although homogeneity in terms of friendship in pairing improves the flow experience, this is not reflected in code performance, and even heterogeneity results in higher code quality. Based on our results, it seemed that the difference in knowledge level between students contributes to the increased effort and learning of the subject. Heterogeneity in prior knowledge may have facilitated knowledge transfer among peers. By supporting this idea, students with lower scores commonly benefit more from programming partnerships, especially in the long term (Smith et al., 2018). Denner et al. (2014) also found that differences in students' prior experience impacted pair programming partnerships. However, Demir and Seferoğlu (2021b) reported that homogeneity or heterogeneity in terms of the level of prior knowledge among peers did not make a difference in terms of group harmony, flow experience, and coding performance. Additionally, the researchers also stated that students with a higher level of prior knowledge are not willing to inform those with a low level of prior knowledge, which in turn results in these students assuming a passive role. Similarly, Bodaker and Rosenberg-Kima (2023) found that students take a more passive role when their partners are more skilled or knowledgeable.

In our study, it was found that the student in the navigator role felt like a passive participant and perceived that the teammate in the driver role learned better. Bodaker and

Rosenberg-Kima (2023) pointed out that this sense of learning (dis)engagement due to role reversal may also arise from differences in knowledge levels. One option to prevent this situation is to focus on role definition. At this point, the role of the navigator as a link between the driver, the teacher, and the other teams can be emphasized. In addition, two computers may also be used in pair programming as a way to ensure more flexible collaboration. Game design and programming activities are inherently open-ended and ill-defined tasks that are open to different solutions. These aspects make them very suitable for demonstrating computational thinking skills from a variety of different perspectives. However, the students emphasized that success criteria should be defined clearly.

Pair programming encourages students to tackle more complex and difficult tasks. Similarly, it has been reported in the literature that the difficulty of the programming tasks has an impact on student collaboration (Xu et al., 2023). Another study stated that as the cognitive load or difficulty level of the task increases, students tend to prefer working in pairs to alleviate intrinsic cognitive load (Zhong & Wang, 2021). Although today's job market is based on a complex software development process that requires collaborative work, programming instruction is traditionally viewed as an individual task and predominantly conducted with screen-based and isolated learning practices (Lai & Wong, 2020). There also appears to be resistance among educators to pair programming, and essentially working with others on the program is often stigmatized by them as cheating (Harsley et al., 2017). The underlying rationale for their bias is due to reservation in the asymmetric distribution of participation and effort between partners in pair programming. It does not seem reasonable for educators to evaluate the pairs' programming performance based on summative assessments or test scores alone, as the students' scores may not be a reliable reflection of their individual efforts (Hahn et al., 2009). Briefly, holistic and comprehensive assessment remains a challenge for educators in evaluating the individual performance and benefits of each partner in a pair.

Conclusion, Limitations, and Implications

This study highlights the high level of student motivation with regard to game programming activities, regardless of the solo or pair programming modality. There was no significant difference in motivation between these programming modalities, although pair programming significantly improved students' problem-solving skills. However, compared to the control group, this improvement is statistically significant only in the area of conceptualizing the problem through modeling. From this standpoint, it can be concluded that pair programming yields more fruitful pedagogical results than solo programming does. Qualitative data also supports this in some ways. The vocal expression of ideas in pair programming not only supports a set of mental models ranging from understanding and analyzing the problem to representing the solution steps, but also allows increased awareness and increased thinking about them. Team dynamics are a primary determinant of learning outcomes. Team compatibility greatly relies on communication and prior knowledge differences between peers, task sharing, and adherence to rules and responsibilities. Game design and programming was found to be an appropriate task type for demonstrating computational skills. These tasks provide students with a highly manipulative and flexible setting for programming.

For practitioners, it seems reasonable to take advantage of the pedagogical outcomes of pair programming in computational thinking education. Especially in the pre-coding process, the verbal expression of thoughts in pair programming helps to model and conceptualize a

problem. Pair programming encourages students to tackle more complex problems. It also reduces teacher dependency and workload. Computer sharing, which seems to be a limitation in settings with inadequate hardware or large class sizes, can be turned into an opportunity. However, the roles of navigator and driver must be clearly defined. Otherwise, the navigator may feel excluded and isolated. It is useful to change partners during the semester, but it is recommended that the period not be too short, such as course hours or assignments. It should be noted that pairing should be voluntary, but that differences in knowledge levels between participants also have a positive effect on team harmony. As the difficulty of the tasks gradually increased, it was observed in our study that the cooperation patterns were more intense in the more challenging tasks. Although game design tasks are suitable for CT skills because they are open-ended and ill-defined, it is critical to clearly define success criteria. More recently, artificial intelligence (AI) software or chatbots have become an option as programming buddies to help students learn and progress better in programming (Groothuijsen et al., 2024; Liu & Li, 2024). However, the learning ecosystem, technical requirements, ethical considerations and opportunities and risks need to be carefully considered.

To understand how students organize their ideas in the pair programming process, how these ideas change during the process, or what stages the students go through, in further studies contextual details about the collaborative problem-solving process can be captured by examining sketches, mind maps, think-aloud processes, flowcharts, and algorithms rather than programming artifacts. In addition, the effectiveness of pair programming can be examined in terms of the communication patterns between pairs. The small sample size in both the quantitative and qualitative parts of our research is a limitation to the generalizability of the findings. Future attempts may be conducted with larger samples.

Contribution Rate of the Researchers

Ö. D., and S. S. S. contributed to the conception and design of the study. Data collection was performed by Ö. D., M. Ç., and Ö. D. performed the data analysis and interpretation of the results. All authors contributed to writing the original draft of the manuscript. They all read and approved the final version of the manuscript.

Statement of Conflict of Interest

The authors declare that they have no conflict of interest.

Data availability statement

The research data analyzed during the current study are available from the corresponding author on reasonable request.

Appendix 1:

Interview Questions

1. What are your thoughts on pair programming or the selection of a partner for coding?
2. What are the qualities of an ideal partner?
3. Have you encountered any problems while pair programming? If so, how did you resolve these problems?
4. How did you divide the tasks and collaborate during pair programming? What form do you think collaboration should take in pair programming?
5. What do you think are the benefits and drawbacks of pair programming? Why do you think these issues arise?
6. What impact has pair programming had on your interest in programming courses or coding? Why do you think this effect occurs?
7. What do you think is the impact of pair programming on your learning of programming?
8. What are your thoughts on the impact of pair programming on the quality of the code you write?

References

Alammary, A. (2019). Blended learning models for introductory programming courses: A systematic review. *PLOS ONE*, *14*(9), e0221765. https://doi.org/10.1371/journal.pone.0221765

Alturki, R. A. (2016). Measuring and improving student performance in an introductory programming course. *Informatics in Education*, *15*(2), 183-204. https://doi.org/10.15388/infedu.2016.10

Baheti, P., Gehringer, E., & Stotts, D. (2002). *Exploring the efficacy of distributed pair programming*. In D. Wells & L. Williams (Eds.), *Extreme programming and agile methods–XP/agile universe 2002* (pp. 208-220). Springer Press. https://doi.org/10.1007/3-540-45672-4_20

Balijepally, V., Mahapatra, R., Nerur, S., & Price, K. H. (2009). Are two heads better than one for software development? The productivity paradox of pair programming. *MIS Quarterly, 33*(1), 91-118. http://dx.doi.org/10.2307/20650280

Bishop-Clark, C., Courte, J., & Howard, E. V. (2006). Programming in pairs with Alice to improve confidence, enjoyment, and achievement. *Journal of Educational Computing Research*, *34*(2), 213-228.

Bodaker, L., & Rosenberg-Kima, R. B. (2023). Online pair-programming: elementary school children learning scratch together online. J*ournal of Research on Technology in Education*, *55*(5), 799-816. https://doi.org/10.1080/15391523.2022.2036653

Bowman, N. A., Jarratt, L., Culver, K. C., & Segre, A. M. (2020). Pair programming in perspective: Effects on persistence, achievement, and equity in computer science. *Journal of Research on Educational Effectiveness, 13*(4), 731-758. https://doi.org/10.1080/19345747.2020.1799464

Chorfi, A., Hedjazi, D., Aouag, S., & Boubiche, D. (2020). Problem-based collaborative learning groupware to improve computer programming skills. *Behaviour & Information Technology, 41*(1), 139–158. https://doi.org/10.1080/0144929X.2020.1795263

Cohen, J. (1988). *Statistical power analysis for the behavioral* sciences (2nd ed.). Erlbaum Press.

Creswell, J. W., & Plano Clark, V. L. (2011). *Designing and conducting mixed methods research*. Sage Press.

Çal, H., & Can, G. (2020). The influence of pair programming on secondary school students' confidence and achievement in computer programming. *Trakya Journal of Education, 10*(1), 221-237. https://doi.org/10.24315/tred.575098

Çınar, M., & Tüzün, H. (2021). Comparison of object-oriented and robot programming activities: The effects of programming modality on student achievement, abstraction, problem solving, and motivation. *Journal of Computer Assisted Learning, 37*(2), 370-386. https://doi.org/10.1111/jcal.12495

Çiftci, S., & Bildiren, A. (2020). The effect of coding courses on the cognitive abilities and problem-solving skills of preschool children. *Computer Science Education, 30*(1), 3-21. https://doi.org/10.1080/08993408.2019.1696169

Demir, Ö., & Seferoğlu, S. S. (2017, October 11-13). Examination of pair programming as a reflection of cooperative problem solving on teaching programming [Paper presentation]. ITTES 2017, İzmir, Türkiye.

Demir, Ö., & Seferoğlu, S. S. (2021a). A comparison of solo and pair programming in terms of flow experience, coding quality, and coding achievement. *Journal of Educational Computing Research, 58*(8), 1448-1466. https://doi.org/10.1177/0735633120949788

Demir, Ö., & Seferoglu, S. S. (2021b). The effect of determining pair programming groups according to various individual difference variables on group compatibility, flow, and coding performance. *Journal of Educational Computing Research, 59*(1), 41-70. https://doi.org/10.1177/0735633120949787

Demirkıran, M. C., & Hocanin, F. T. (2021). An investigation on primary school students' dispositions towards programming with game-based learning. *Education and Information Technologies*, *26*(4), 3871-3892. https://doi.org/10.1007/s10639-021-10430-5

Denner, J., Green, E., & Campe, S. (2021). Learning to program in middle school: How pair programming helps and hinders intrepid exploration. *Journal of the Learning Sciences, 30*(4-5), 611-645. https://doi.org/10.1080/10508406.2021.1939028

Denner, J., Werner, L., Campe, S., & Ortiz, E. (2014). Pair programming: Under what conditions is it advantageous for middle school students? *Journal of Research on Technology in Education, 46*(3), 277-296. https://doi.org/10.1080/15391523.2014.888272

Dybå, T., Arisholm, E., Sjoberg, D. I. K., Hannay, J. E., & Shull, F. (2007). Are two heads better than one? On the effectiveness of pair programming. *IEEE Software, 24*(6), 12-15. https://doi.org/10.1109/MS.2007.158

Fanchamps, N., Slangen, L., Hennissen, P., & Specht, M. (2021). The influence of SRA programming on algorithmic thinking and self-efficacy using Lego robotics in two types of instruction. *International Journal of Technology and Design Education*, *31*(2), 203-222. https://doi.org/10.1007/s10798-019-09559-9

Fraenkel, J. R., Wallen, N. E., & Hyun, H. H. (2012). *How to design and evaluate research in education* (*8th ed.)*. The McGraw-Hill Press.

Groothuijsen, S., van den Beemt, A., Remmers, J. C., & van Meeuwen, L. W. (2024). AI chatbots in programming education: Students' use in a scientific computing course and consequences for learning. *Computers and Education: Artificial Intelligence*, *7*, Article e100290. https://doi.org/10.1016/j.caeai.2024.100290

Hahn, J. H., Mentz, E., & Meyer, L. (2009). Assessment strategies for pair programming. *Journal of Information Technology Education-Research*, *8*, 273-284.

Hanks, B., Fitzgerald, S., McCauley, R., Murphy, L., & Zander, C. (2011). Pair programming in education: A literature review. *Computer Science Education, 21*(2), 135-173. https://doi.org/10.1080/08993408.2011.579808

Harsley, R., Fossati, D., Di Eugenio, B., Green, N. (2017, March 8). *Interactions of individual and pair programmers with an intelligent tutoring system for computer science* [Paper presentation]. 2017 ACM SIGCSE Technical Symposium on Computer Science Education, Seattle, Washington, USA. https://doi.org/10.1145/3017680.3017786

Hawlitschek, A., Berndt, S., & Schulz, S. (2023). Empirical research on pair programming in higher education: A literature review. *Computer Science Education*, *33*(3), 400-428. https://doi.org/10.1080/08993408.2022.2039504

Jonassen, D. H. (2000). Toward a design theory of problem solving. *Educational Technology Research and Development*, *48*(4), 63-85. https://doi.org/10.1007/BF02300500

Kalelioglu, 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.

Kazimoglu, C., Kiernan, M., Bacon, L., & Mackinnon, L. (2012). A serious game for developing computational thinking and learning introductory computer programming. *Procedia - Social and Behavioral Sciences, 47*, 1991-1999. https://doi.org/10.1016/j.sbspro.2012.06.938

Keller, J. M. (1987). *IMMS: Instructional materials motivation survey.* Florida State University Press.

Krizsan, T., & Lambic, D. (2024). Examining the impact of pair programming on efficiency, motivation, and stress among students of different skills and abilities in lower grades in elementary schools. *Education and Information Technologies.* https://doi.org/10.1007/s10639-024-12859-w

Kutu, H., & Sözbilir, M. (2011). Adaptation of instructional materials motivation survey to Turkish: A validity and reliability study. *Necatibey Faculty of Education Electronic Journal of Science & Mathematics Education, 5*(1), 292-312

Lai, X. Y., & Wong, G. K. W. (2022). Collaborative versus individual problem solving in computational thinking through programming: A meta-analysis. *British Journal of Educational Technology*, *53*(1), 150-170. https://doi.org/10.1111/bjet.13157

Liu, J., & Li, S. (2024). Toward artificial intelligence-human paired programming: A review of the educational applications and research on artificial intelligence code-generation tools. *Journal of Educational Computing Research*, *62*(5), 1385-1415. https://doi.org/10.1177/07356331241240460

Lubarda, M. V., Phan, A. M., Schurgers, C., Delson, N., Ghazinejad, M., Baghdadchi, S., . . . Qi, H. (2024). Virtual pair programming and online oral exams: effects on social interaction, performance, and academic integrity in a remote computer programming course. *Computer Science Education*, 1-41. https://doi.org/10.1080/08993408.2024.2344401

Lussier, R. N. (1996). Human relations in organizations: A skill-building approach. Irwin Press.

Mazman, S. G. (2013). *Modeling the influence of cognitive based individual differences on programming performance* (Unpublished doctoral dissertation). Hacettepe University, Institute of Educational Sciences, Ankara, Türkiye.

Mehta, K., & Sood, V. M. (2023). Agile software development in the digital world – Trends and challenges. In S. Hooda, V. M. Sood, Y. Singh, S. Dalal, & M. Sood (Eds.), *Agile software development* (pp. 1-22). Scrivener Publishing-Wiley Press.

Navarro-Cota, C., Molina, A. I., Redondo, M. A., & Lacave, C. (2025). Individual differences in computer programming: a systematic review. *Behaviour & Information Technology, 44(*2), 357-375. https://doi.org/10.1080/0144929X.2024.2317377

Noone, M., & Mooney, A. (2018). Visual and textual programming languages: a systematic review of the literature. *Journal of Computers in Education, 5*(2), 149-174. https://doi.org/10.1007/s40692-018-0101-5

Pilkington, C. (2018). A playful approach to fostering motivation in a distance education computer programming course: Behaviour change and student perceptions. *International Review of Research in Open and Distributed Learning, 19*(3), 282-298. http://dx.doi.org/10.19173/irrodl.v19i3.3664

Plonka, L., Sharp, H., & van der Linden, J. (2012). *Disengagement in pair programming: Does it matter?* In M. Glinz, G. Murphy, & M. Pezze (Eds.), 34th International Conference on Software Engineering (pp. 496-506). IEEE Press. http://dx.doi.org/10.1109/ICSE.2012.6227166

Psycharis, S., & Kallia, M. (2017). The effects of computer programming on high school students' reasoning skills and mathematical self-efficacy and problem solving. *Instructional Science*, *45*(5), 583-602. https://doi.org/10.1007/s11251-017-9421-5

Ramalingam, V., & Wiedenbeck, S. (1998). Development and validation of scores on a computer programming self-efficacy scale and group analyses of novice programmer self-efficacy. *Journal of Educational Computing Research, 19*(4) 365-379. https://doi.org/10.2190%2FC670-Y3C8-LTJ1-CT3P

Razzouk, R., & Shute, V. (2012). What is design thinking and why is it important? *Review of Educational Research, 82*(3), 330-348. https://doi.org/10.3102/0034654312457429

Smith, M. O., Giugliano, A., & DeOrio, A. (2018). Long term effects of pair programming. *IEEE Transactions on Education*, *61*(3), 187-194. https://doi.org/10.1109/TE.2017.2773024

Strauss, A., & Corbin, J. (1990). *Basics of qualitative research*. Sage Press.

Tan, J. B., Wu, L., & Ma, S. S. (2024). Collaborative dialogue patterns of pair programming and their impact on programming self-efficacy and coding performance. *British Journal of Educational Technology*, *55*(3). https://doi.org/10.1111/bjet.13412

Tsai, C. Y. (2019). Improving students' understanding of basic programming concepts through visual programming language: The role of self-efficacy. *Computers in Human Behavior*, *95*, 224-232. https://doi.org/10.1016/j.chb.2018.11.038

Tsompanoudi, D., Satratzemi, M., & Xinogalos, S. (2015). Distributed pair programming using collaboration scripts: An educational system and initial results. *Informatics in Education, 14*(2), 291-314. https://doi.org/10.15388/infedu.2015.17

Wing, J. M. (2006). Computational thinking. *Communications of the ACM, 49*(3), 33-35. https://doi.org/10.1145/1118178.1118215

Xu, F., & Correia, A. -P. (2024). Adopting distributed pair programming as an effective team learning activity: a systematic review. *Journal of Computing in Higher Education, 36*(2), 320-349. https://doi.org/10.1007/s12528-023-09356-3

Xu, W., Wu, Y., & Ouyang, F. (2023). Multimodal learning analytics of collaborative patterns during pair programming in higher education. *International Journal of Educational Technology in Higher Education*, *20*(1), 8. https://doi.org/10.1186/s41239-022-00377-z

Yaman, S., & Dede, Y. (2008). A scale for adults' problem solving skills. *Journal of Educational Sciences & Practices*, *7*(14), 251-269

Yang, Y. F., Lee, C. I., & Chang, C. K. (2016). Learning motivation and retention effects of pair programming in data structures courses. *Education for Information, 32*(3), 249-267. https://doi.org/10.3233/efi-160976

Zarb, M., & Hughes, J. (2015). Breaking the communication barrier: Guidelines to aid communication within pair programming. *Computer Science Education, 25*(2), 120-151. https://doi.org/10.1080/08993408.2015.1033125

Zhong, B., & Wang, J. (2021). Exploring the non-significant difference on students' cognitive load imposed by robotics tasks in pair learning. *International Journal of Social Robotics, 14*, 3-13. https://doi.org/10.1007/s12369-021-00764-y

Zhong, B. C., & Li, T. T. (2020). Can pair learning improve students' troubleshooting performance in robotics education? *Journal of Educational Computing Research, 58*(1), 220-248. https://doi.org/10.1177/0735633119829191