IJCSES

Volume 3, No: 3

January 2020

ISSN 2513-8359

International Journal of Computer Science Education In Schools

Editors

Prof. Filiz Kalelioglu

Yasemin Allsop

www.ijcses.org

International Journal of Computer Science Education in Schools

January 2020, Vol 3, No 3

DOI: https://doi.org/10.21585/ijcses.v3i3

Table of Contents

	Page
Ünal Çakıroglu[1], Samet Atabas[2], Dogukan Sarıyalçınkaya[3], İbrahim Enes Öner[1] Learning programming online: Influences of various types of feedback on programming performances	3- 18
Pınar Mıhcı Türker[1], Ferhat Kadir Pala[1] The Effect of Algorithm Education on Students' Computer Programming Self-Efficacy Perceptions and Computational Thinking Skills	19 - 32

Learning programming online: Influences of various types of feedback on programming performances

Ünal Çakıroglu[1]
Samet Atabas[2]
Dogukan Sarıyalçınkaya[3]
İbrahim Enes Öner[1]

[1]Trabzon University, Faculty of Education
[2]Avrasya University, Vocational School
[3]Ondokuz Mayıs University, Vezirköprü Vocational School

DOI: **10.21585/ijcses.v3i3.57**

Abstract

This article draws on a one-semester online programming course to determine the relationships between the types of feedback that students received and their programming performances. An explorative study was designed including both quantitative and qualitative data. Participants were 15 second-year students enrolled at an Information Technology department of a public university who received weekly programming problems as projects. The instructor provided various types of feedback (corrective, confirmatory, explanatory, diagnostic and expanded feedback) to the students via Github. The results showed that; when the students who received corrective and explanatory feedback more, the programming performance scores were high. A strong positive relationship existed between a total amount of feedback with programming performance scores, also a number of times that students received explanatory feedback and programming performance scores. In addition, while the positive correlations existed between programming performance scores and the amount of all types of feedback, there was a weak correlation between the number of times that students received expanded feedback and programming performance scores. The difficulty of the programming projects, the needs of the participants and the affordances of the online learning environment played key role in the students' preference for receiving feedback type. Recommendations for future research and practice are included.

Keywords: types of feedback, online programming course, programming performances

1. Introduction

In recent years, the increasing demand for programming has led many institutions to utilize online settings for delivering programming courses. Despite the advance in the tools for programming instruction, teaching

programming remains a big challenge for instructors and programming courses are still considered complex and often resulting in low retention (Barr & Guzdial, 2015; Sáez-López, Román-González, & Vázquez-Cano, 2016). Especially, in online programming courses, repetitive failures during lessons may lead students to lose enthusiasm and interest (Law, Lee, & Yu, 2010). In this sense, students often need feedback from the instructors to perform programming tasks (Eng, Ibrahim, & Shamsuddin, 2015). Through ongoing monitoring of online learning, instructors can observe students' progress and guide them via feedback. Effective feedback can support students to progressively identify their strengths and weaknesses and refine their understanding (Gikandi, Morrow, & Davis, 2011).

Researchers in online learning argues that online learning has distinct pedagogical demands owing to the asynchronous nature of interactivity between the teacher and learners (Naidu, 2007). Thus, although online feedback has the potentials to enhance online learning outcomes, the process of using feedback online may be somewhat different from the face-to-face settings. The instructors giving and the students' receiving process may be affected by various factors of the online learning components. This can also affect the preferences of various types of feedback. Thus, a question may come into mind: "Does use of different types of feedback affect the learning outcomes differently?" Although feedback in teaching online programming have a positive effect on outcomes, the relationship between the feedback types used and the programming performances during the instructional process need to be investigated further.

1.1. Feedback in Online Programing

In online learning, the responsibility of learning is shifted from the instructor to the student (Xia & Liitiäinen, 2017). Researchers argued that traditional pedagogical practices that do not fit online classrooms (Baran, Correia, & Thompson, 2011). In this line, ongoing support for scaffolding is suggested in online learning, and the learning process can be facilitated through well-designed feedback (Ludwig-Hardman & Dunlap, 2003). According to Hattie and Timperley (2007), the feedback has been shown to hold great potential for student learning in higher education. The instructor may increase student motivation to learn by facilitating self-directed studies with well-organized instructions and timely feedback (Kop, 2011; Xia, 2015). Feedback enhances learning by informing the user about learners' performance, correcting the user, explaining the correct answers, evaluating, motivating the user, rewarding the user or attracting attention (Lawless & Pellegrino, 2007).

Creating pedagogically sound feedback is a complex task. Feedback content, learning tasks, characteristics of the students, frequency of the feedback and timing may affect to contributions of the feedback to the learning outcomes (Kluger & DeNisi, 1996; Shute, 2008). In this regard, divergent views exist about what to include in feedback messages, and even in the amount of information to reach the pedagogical objectives (Campos, et al., 2012). Instructors should have prior knowledge about students' readiness, the sources of the mistakes to provide the appropriate type of feedback to meet the students' needs.

1.2. Types of Feedback

In online learning giving and receiving online feedback is somewhat different than the traditional feedback, because learners have more time to deal with feedback in the tasks (Pyke & Sherlock, 2010). Considering the online learning settings, Vasilyeva et al. (2007) classified the functions of online feedback as no feedback, simple verification feedback, correct response feedback, elaborated feedback and try again feedback.

Schimmel (1983) classified types of feedback as explanatory, corrective, diagnostic, confirmatory and expanded.

Explanatory feedback gives information to the students about their learning outcomes including why

the answer is wrong or correct.

Corrective feedback provides the correct answer which is given along with the confirmatory feedback through the student's learning outcomes.

Diagnostic feedback includes information about what the student should work to correct the wrong answer and how it should work.

Confirmatory feedback provides information about the results of learning whether it is true or false.

Expanded feedback enables the student to expand students' existing knowledge by helping them to build relationships between prior knowledge and new knowledge.

Research conducted about online feedback includes different perspectives. For instance, Gikandia and Marrow (2016) used Moodle and found that peer formative feedback was useful in understanding how the students learned actively and contributed to peer review. Cakiroglu et al. (2016) studied the effects of instant feedback in an online programming course and observed that the online course cultivated students' programming skills. In another study Horne at all. (2018) indicated that it is beneficial for students to receive feedback on a regular basis, but that they do not need to use excessive feedback to get a positive impact. While prior work provides compelling evidence that feedback is one of the keys to successful programming, it leaves several open questions about the effect of types of feedback. Following conclusions from the previous studies, we aim at gaining an insight into the nature of feedback types provided in online programming instruction.

In an online learning environment, various tools can be used for giving feedback. LMSs are common settings used to deliver online settings. Also, some specialized web-based tools are used in online learning. For instance,, GitHub is originally developed as a tracking software development and collaboration during the software development process. GitHub provides educational tools such as GitHub Classroom and GitHub Education. Thanks to these tools GitHub fulfills some requirements of LMS, such as asynchronous communication, immediate feedback, content delivering and evaluation. The Fork provides to copy the repositories belonging to someone else. Anyone in the GitHub group can develop projects and receive visual feedback with graphs of all commits and fork made.

1.3. Need for Study

Like all learning models, online learning has some inherent problems, especially in the areas of isolation, support, technology, discipline and feedback (Mishra & Koehler, 2006). It is valuable for instructors to acknowledge the importance of students' needs and expectations in formulating their feedback (Hyland, 2010; Jara & Mellar, 2010). Although previous research has investigated widespread use of feedback in various courses, the conditions under which types of feedback foster learning online programming remain unclear. Thus, the findings of this study will be useful to guide for instructors to provide appropriate feedback types for the context.

1.4. Purpose of the Study

Given the importance of feedback for learning programming and the increasing trend of online programming courses, this study seeks to examine which types of feedback enhance students' success in the given tasks in the online training environment. Particularly, Github as an asynchronous programming teaching platform is used.
In line with the overall purpose of the study, the following research questions were directed:

- How did different types of feedback effect on programming performance scores?
- How did students explain the contribution of various feedback types to the online programming learning process?

2. Method

In order to address the research questions, an explorative study was designed. One semester online course was carried out through synchronous online techniques. Both quantitative and qualitative data were used gathered to understand the relationships between the types of feedback and programming performances. The instructor gave one project per week for 9 weeks through GitHub, and students worked projects on Github by receiving feedback within this platform.

GitHub was developed to track the development of projects, share them with other developers and to provide a common development environment. The affordances of Github in the collaboration and feedback facilities directed programmers to use it for programming instruction (Gunnarsson, et al., 2017). GitHub fulfills a lot of features of an LMS, such as asynchronous communication, immediate feedback, transmitting content and evaluate students. For instance, Fork provides to copy the repositories belonging to someone else. Visual feedback provides the users with graphs of all commits and fork made. So that anyone to follow the project during its development.

2.1. Participants

In this study, participants were 15 (12 male, 3 female) students (between 20-29 age) enrolled in a programming department of a vocational high school in Turkey. The study was carried out in Javascript class in the 4th semester of the academic year 2017–2018.

2.2. Process

The course was delivered via both synchronous online sessions via Adobe Connect 3 hours per week. In these sessions, the instructor generally delivered presentations and sample programming codes and provided discussions about the codes. Also, the instructor asked students to work on 9 programming projects during a semester of 14 weeks. Projects covering conceptual and strategic knowledge of programming were presented starting from easy to difficult projects. Projects include tasks about the programming knowledge that is provided by the instructor during the week. Students were required to complete the online tasks in their own learning time at home. The instructor gives the projects on GitHub and students sent the answers including design of the form, the question, the answer screenshots about the answer and detailed explanations are provided by the GitHub Repo. The projects are briefly presented in Table1.

Table 1. Weekly programming projects

Projects	Programming Concepts	Project topics
P1	Basic variable and transaction	Priority of operations
P2	Basic loop and comparison	Divisibility question
P3	Inner loop	Multiplication table of numbers
P4	Conditions (if-else)	Odd or even numbers
P5	Usage of function	Environment, area, volume
P6	Logical testing and loop	Calculation of multiplication
P7	Arrays	Sorting the numbers
P8	Operators	Rules about triangles
P9	Number guessing game	Random numbers

The instructor examined the programming codes and provided feedback to the students' programming codes on the GitHub Commit tool. Using the repository of GitHub students were allowed to submit their programming codes for the projects. Students could follow the feedback for all projects. In the feedback, the instructor provided conceptual and logical information about programming. Also, the structure of Javascript and the web page design was also included in the feedback. The instructor was able to follow students' progression in the tasks via GitHub. Using their own accounts in the Fork tool, students provided their codes and their responses for the feedback including how they benefited from feedback. At the end of the deadline for each project, students submitted the final version of programming codes and shared them with other students via GitHub.

The answers for the programming projects were shared by the instructor at the end of the project deadline. A view from the project including the student response and the instructor feedback is presented in Figure 1.

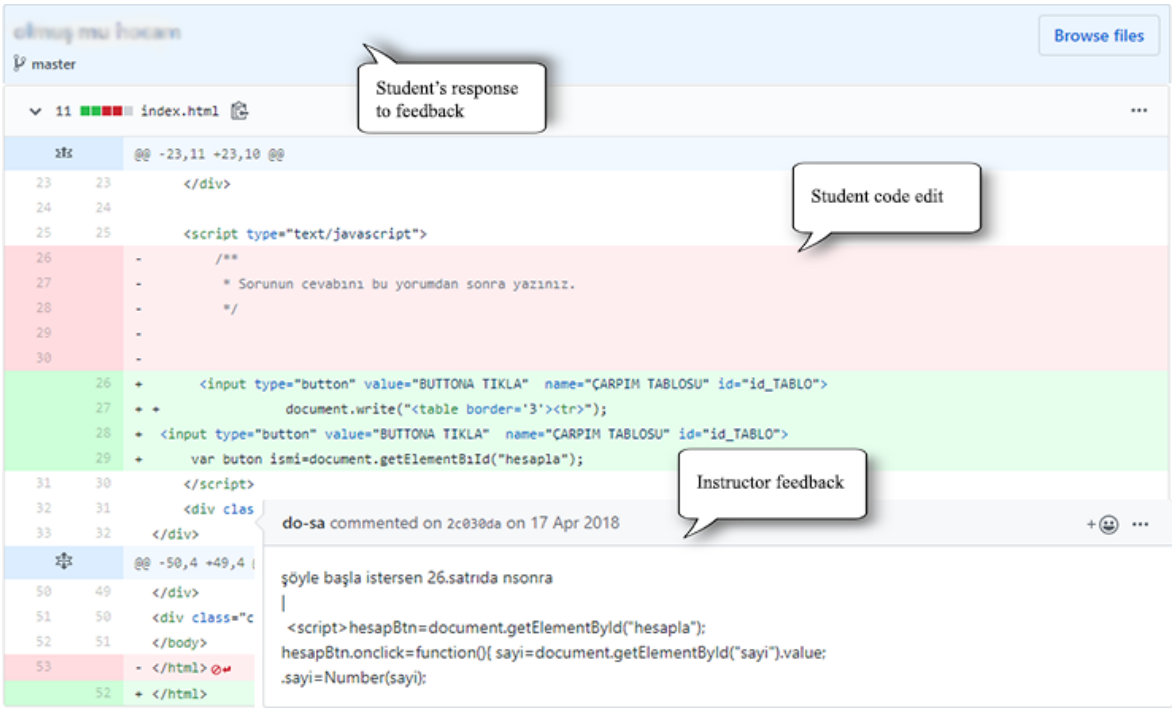

Figure 1. A view from the journey of a feedback

2.3. Data Collection Tools and Analysis

Both quantitative and qualitative data were used to acquire a better understanding of the influences of feedback on students' programming performances. Data Collection tools include Github logs were analyzed as quantitative data collection tools, also rubric and question form were used to help explain the quantitative results from Github data.

GitHub logs: GitHub logs included quantitative data (amount and time) about students' answers for the project tasks and instructor feedback. The feedback is given and the students' responses for the feedback (their revisions regarding the feedback or explanations about the feedback) were provided through GitHub. The logs were analyzed considering the amount of types of feedback. Two researchers determined the types of feedback on the projects.

Rubrics: An evaluation rubric was created for examining students' programming codes in order to determine their performances. Rubrics include an evaluation protocol for three types of programming knowledge (syntactic knowledge, conceptual and strategic knowledge) (McGill, & Volet, 1997) in their tasks. Two experts in online

learning and programming instruction reviewed the rubric items and the scales for content validity. Three sections (syntax, conceptual, strategic) were grouped in three levels of evaluation as regarding the total scores of 0-15. The T- scores of the total scores from the rubric was taken as a performance score. Two researchers first assigned the scores for the tasks on the rubric individually, and then they discussed the students' codes each other until they come to reach an agreement on the score of students for the task. The well-defined score categories assisted in maintaining consistent scoring when rating. Spearman Correlation tests were used to determine the relationships between the amount of each type of feedback used during the instructional process.

According to Schimmel (1988), feedback differs according to the amount of information its solution to the wrong answer (except for the last step). In this study, feedback given by teachers in the learning-teaching process was handled according to Schimmel's classification of feedback types that provide information to students and based on the amount of this information. Answers such as true, false, yes, no are confirmative feedback. The corrective feedback provides the correct response to the student. For example, "Wrong answer, Turkey's capital is Ankara." There are various forms of use of descriptive feedback. It can be presented in stages by showing the content to the student before giving the wrong answer. The diagnostic feedback aims to correct the student's erroneous mental modeling process. Therefore, the answer has more function than just correcting it. The expanded feedback was considered feedback directing the student to find the result own.

Open-Ended Question Form: The form includes open-ended questions about feedback, difficulties in the process and effects of the feedback on the learning progress and closed-ended questions for demographics. The form was given to the participants as a result of the training and the form was completed themselves. The form included such as "What were the challenges you faced during the process?", "How did the comments written to you on GitHub affect your learning of programming?". "What kind of feedback did you benefited most? Why did you think that you benefited more?"

3. Results

The programming performances were evaluated by scoring the students' responses to the tasks in the weekly projects through rubrics. Students' perspectives in the question form were used to explain the contributions of the feedback to the programming performance.

3.1. Relationships between Feedback Students Received and Programming Performances

In order to determine the relationships between the feedback types and the programming performances, first, we defined the relationships descriptively on the basis of all students and all projects. Then we provided statistical results to understand this relationship.

3.1.1. Amount of Feedback and Programming Performances

The amount of feedback was differentiated in the projects due to the students' needs. We considered the amount of feedback as the indicators of feedback use. Totally 292 posts were provided by students through 547 feedback in the types of corrective, confirmatory, explanatory, diagnostic and expanded feedback. The programming performance scores, and the total amount of feedback were presented together in Figure 2.

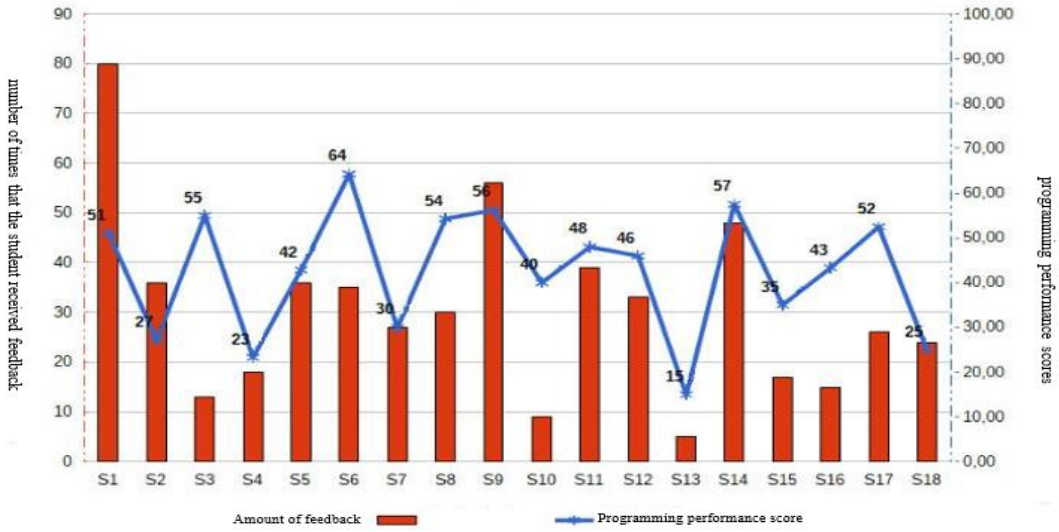

Figure 2. Relations between the amount of feedback (per student) and programming performances

In the question forms students also expressed that they were highly satisfied with receiving feedback as part of learning programming. Some of the participants indicated that feedback was helpful, reminding them that they were guiding them in correcting their mistakes. In this sense, S10 stated that: "*The comments my teacher provided helped me to define the correct programming statements. This was a good way that teacher's comments directed us when we try to program*."

3.1.2. Types of Feedback and Programming Performances

Totally 547 feedback including 157 corrective, 53 confirmatory, 204 explanatory, 100 diagnostic and 33 expanded feedback were used in 9 projects. The average of the feedback for each project is 60,7. Figure 3 shows the total amount of feedback and average programming performance scores. While students' average scores were 51, they received an average of 42 feedback for each project. It was found that, the student received a lot of feedback, the programming performance scores were generally high. As a matter of fact, three students (S9, S14) who received the much amount feedback got the highest scores. It was worth noting that the average score of some students was higher than the other students even though they receive a low amount of feedback. For instance, S10 and S3 received a low amount of feedback, but their programming performance scores were 40, 55 respectively. Likewise, S6 who got the highest average score (64.29) from the projects, received averagely 35 feedback from all projects. It was seen that 61% of the students had a score above the average, while others were at the interval of 15 to 40. At this point, in order to determine whether the types of feedback were influenced by the programming performances much, a weekly analysis of the feedback was provided. Descriptively, the results based on the projects were presented in Figure 3.

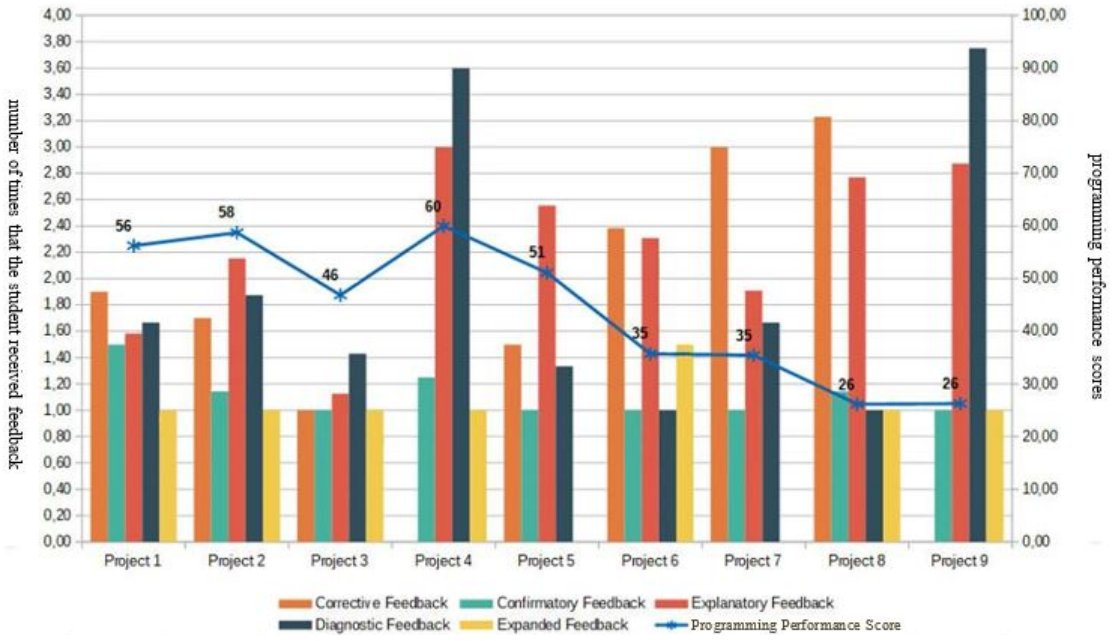

Figure 3. Relations between the number of times that students received feedback (per project) and programming performance scores

Figure 3 shows that the most frequently used feedback types were explanatory, corrective and diagnostic feedback. While more amount of diagnostic feedback was given in Project 4 and Project 9, also more amount of explanatory feedback was provided in Project 7 and Project 8. Also, the instructor provided an average amount of the corrective feedback high. The amount of expanded feedback that the student received was less for each project. Although the amount of feedback use increases towards the end of the implementation, it seems that the average of the scores of the students' answers decreased gradually.

3.1.3. Correlations between the Amount of Feedback and Programming Performances

In addition, statistically, Table2 shows the relationship between the amount of feedback that the students received and programming performances. If the sample size is less than 30, data distribution is not normal or heterogeneous data structures, the Spearman correlation analysis method is used (Bishara, & Hittner, 2012). As can be seen from the table, the Spearman Product Moment Correlation analysis, which was conducted to determine the relationship between the rubric scores obtained from each project and the total amount of feedback in each project. Statistically, the correlation coefficient was interpreted with its value as below:

Exactly -1. A perfect (negative) linear relationship, -0.70. A strong (negative) linear relationship, -0.50. A moderate (negative) relationship, -0.30. A weak (negative) linear relationship,0. No linear relationship, +0.30. A weak (positive) linear relationship, +0.50. A moderate (positive) relationship, +0.70. A strong (positive) linear relationship, Exactly +1. A perfect (positive) linear relationship.

The results indicate that higher programming performance scores positively correlate with more feedback (gathered from the rubric) (<.05, r=0.884). The correlation between the programming performance scores obtained from each project and the total amount of corrective feedback in each project correlates moderate positive at p <.05 level (r=0.615). The higher programming performance scores also correlates with more confirmatory

feedback moderate positively (r=0.675). Also, the programming performance scores and the amount of explanatory feedback that the students received showed a moderate positive correlation (r=0.771). Moderate positive correlation also exists between the number of items of diagnostic feedback and programming performance scores (r=0.519). Weak positive correlation exits only between high programming performance scores and more expanded feedback (r=0.261).

Table 2. Correlations between number of times that the students received feedback and programming performances

		The number of times that the students received						
Correlations Spearman's rho		total	corrective feedback	confirmative feedback	explanatory feedback	diagnostic feedback	expanded feedback	programming performance scores
Total	CC	1,000	,692**	,678**	,911**	,585**	,329**	,884**
	Sig. (2-tailed)		,000	,000	,000	,000	,000	,000
	N	162	162	162	162	162	162	162
Corrective feedback	CC	,692**	1,000	,444**	,643**	,002	,010	,615**
	Sig. (2-tailed)	,000		,000	,000	,978	,895	,000
	N	162	162	162	162	162	162	162
Confirmative feedback	CC	,678**	,444**	1,000	,485**	,275**	,143	,675**
	Sig. (2-tailed)	,000	,000		,000	,000	,070	,000
	N	162	162	162	162	162	162	162
Explanatory feedback	CC	,911**	,643**	,485**	1,000	,476**	,177*	,771**
	Sig. (2-tailed)	,000	,000	,000		,000	,024	,000
	N	162	162	162	162	162	162	162
Diagnostic feedback	CC	,585**	,002	,275**	,476**	1,000	,391**	,519**
	Sig. (2-tailed)	,000	,000	,000	,000		,000	,000
	N	162	162	162	162	162	162	162
Expanded feedback	CC	,329**	,010	,143	,177*	,391**	1,000	,261**
	Sig. (2-tailed)	,000	,895	,070	,024	,000		,001
	N	162	162	162	162	162	162	162
Programming performance scores	CC	,884**	,615**	,675**	,771**	,519**	,261**	1,000
	Sig. (2-tailed)	,000	,000	,000	,000	,000	001	
	N	162	162	162	162	162	162	162

The number of times that the students received

**Correlation is significant at the 0.01 level (2-tailed) CC: Correlation Coefficient

* Correlation is significant at the 0.05 level (2-tailed)

3.2. Contributions of Various Feedback to the Programming Learning Process

Considering the types of feedback, most of the students evaluated the corrective feedback helpful in their problem-solving process. In this sense, S1 stated: *"I think the feedback facilitated my work because they showed me my mistakes and how can I correct them".* Another student S2 also emphasized the motivation effect of corrective feedback as *"Sometimes I forgot how to use the statements and I could not continue on writing the code, at that time, instructor comments made me see my mistakes and continue writing".* Related with the tasks in project 6 and project 8 (requiring extra programming conceptual knowledge about the structure of the language), a few numbers of students emphasized the contribution of the expanded feedback. For instance, according to S4, the instructor suggested them various kinds of resources; and taking advantage of these, they could investigate some structural and logical knowledge to write effective and correct codes. Similarly, S5 explained that instructor feedback for searching other sources helped them not only writing correct codes but also learn why to use such code pieces or functions. In this regard, S6 stated that various examples from various programming web sites which the instructor suggested were very helpful. Some of the students also indicated the effectiveness of the diagnostic feedback. They identified that this kind of feedback was able to make them aware of the mistakes in the programs. In this line, S10 stated: "*Sometimes I have little mistakes in the codes but I cannot find where it is. Thanks to instructors' comments that he shows us where the mistake is. Sometimes I consider it as syntax error but the instructor shows that it was about the programming logical structure*.". Besides, most of the participants claimed that feedback was needed to be more explanatory.

It was observed that the first responses to the tasks (codes for the problems) were including more mistakes, and students needed more descriptive feedback, namely including some explanations about the way for solutions. Some of them addressed that, they needed some basic examples directing them to define the ways of solutions for the weekly problems. Thus, they considered that more explanatory tips would have been more helpful than corrective feedback. In addition, students' perspectives showed that in some complex tasks more than one type of feedback was required to be given together. Surprisingly, depending on the students' prior programming knowledge, when the prior knowledge is low, the expanded feedback was not sufficient for their needs and they need explanatory feedback within the expanded one.

On the other hand, students' perspectives showed that the complexity of the problems was a crucial factor for using a diversity of feedback. For instance, S6 expressed that after starting the coding in complex problems she often could not continue, because she did not understand the tasks that the instructor asks them to do. According to S6, expanded feedback was not enough solely, so the instructor also added corrective feedback for some tasks.

4. Discussion and Conclusions

4.1. Relationships between Types of Feedback and Programming Performances

Hyland (2001) pointed out that online learners receive feedback from the online instructor is a useful means of communication between the students and the instructor. In this study, we focused on the contributions of using different types of feedback to the programming performances. The results indicated that various feedback in online programming instruction have a positive role in facilitating students' learning programming. In accordance with the findings, another research project addressed that during the programming process, students need to be aware of their mistakes by efficient feedback (Hatziapostolou & Paraskakis, 2010). Although some students' log data confirm the idea that a positive correlation exists between more feedback and programming performance scores,

it was not supported by the data of a few numbers of students. Statistically, a strong correlation (r=0.884) was found between the rubric scores and the total amount of feedback. Although students in higher programming performance scores did not confirm that higher score correlates with more feedback (Niessen, Meijer, & Tendeiro, 2016); the adverse relationship was confirmed that is when the amount of the feedback received was less, the programming performance scores are also low. It was also found that the amount of feedback for each project was quite more in difficult projects. One reason for the might be that the nature of programming tasks influences the amount of feedback. Independent from types of feedback, the relationship between total number of times that the student received feedback and the programming performance scores was positive. The results were consistent with the study that the coding success of first-year students who received programming lessons through getting feedback during the course was high (Benachour & Edwards, 2009). The present study is also consistent with the findings of Kyrilov (2017) who examined the effectiveness of the feedback in programming exercises with the help of code texts and forms. As a result, feedback received by the students contributed positively to their programming performances. Conversely, inconsistent with the results of the present study, (Heo & Chow (2005) investigated the effectiveness of feedback on students' programming performances and showed that there was no benefit of feedback in online learning.

In this study, some types of feedback were given more than the others. It was observed that the frequency of use of feedback types was in the following order: explanatory, corrective, diagnostic, expanded and confirmatory in this study. One reason for the highest use of explanatory feedback may be the way of the interaction between students and instructors on the GitHub. As a matter of fact, the instructor could improve explanations in each of the posts when the participants require it. One other reason may be students' previous learning experiences. Because some of the students were accounted to be informed about their responses and took the explanations into consideration during the learning process.

4.2. Contributions of the Feedback to the Learning Process

According to Orsmond & Merry (2011), active use of feedback does not seem to be the primary choice for many students. In this study, the expanded feedback was used less because using this type of feedback does not cover enough information for the students to provide solutions for the programming projects. While expanded feedback includes useful knowledge for the given problems, sometimes students need more time to distill the required knowledge. Another obstacle to using this kind of feedback may be the lack of know-how to use this feedback. In this regard, researchers argue that, when the students could not apply strategies for using feedback for their tasks, they gave up using that feedback. A problem, however, is that in order to be effective, feedback should not only be delivered appropriately, but it should also be taken into consideration by students (Nicol & Macfarlane-Dick, 2006). Therefore, students sometimes needed explanatory feedback in order to use the feedback to infer the conceptual knowledge required for their projects.

Overall, one reason for the positive effect of feedback on the programming performances may be the voluntariness of user. In this study, students did not have to get support from feedback. Accordingly, they did not provide extra effort to think about what they can do with the feedback. Similarly, Zaini (2018) addressed that when students considered the feedback compulsory and in turn altered their voice, they cannot benefit from the feedback as expected. The findings from open-ended questions reflect that the way of using feedback in this study provided a motivational influence on continuing to write programming codes and correcting the mistakes during the coding journey. One reason for this might be that; some of the students' considered that the feedback provided on time when they had problems while writing codes and the projects were not interrupted. Accordingly, their interest

increased to continue writing the code.

On the other hand, the difficulty of the projects might be another factor influenced the feedback type given to the students. Because the students' responses to the difficult projects were not the same as those of the easy projects. In this study, the projects were given in an easy to difficult manner. In the easy projects, the expanded feedback was given less and the confirmatory feedback was given more. This was because most of the participants could do easy projects without referring to feedback.

The confirmation that syntax or some basic conceptual knowledge generally met students' needs in which they did not need more coding clues in the easy projects at the beginning of the instructional process. When most of the students could provide correct solutions for the projects, the instructor could not diagnose accurately the type of feedback that the students need. In this sense, the diagnostic feedback was not used more. In the advanced projects, the instructor provided more diagnostic feedback and expanded feedback. In the first five projects, the high mean programming performance scores somewhat positively correlates with more feedback, but the relationships in the subsequent projects which include difficult tasks projects are not clear. In the first five projects, the instructor asked the participants to provide solutions by themselves, thus the feedback was mostly corrective and explanatory. In the last five projects, the instructor and the participants interacted more than those of the first five projects. The instructor provided expanded feedback with the explanatory feedback together when it was not enough for the participants to understand the tips. At this time, sometimes he fulfilled shortcomings in students' code pieces to help them.

In this study, students' perspectives about the feedback in the projects showed that they preferred the feedback mostly when they were useful for them. It is remarkable that students mostly prefer specific and individualized feedback. Similarly, Walker (2009) found that if the students were engaged in one particular assignment, they want feedback directly appealing to them. However, providing feedback for all students individually in an online programming course is a time-consuming process (Duffy & Kirkley, 2004). In this sense, the instructors' role had a crucial influence in this study to provide feedback as soon as possible and to post responses for students' questions. It can be thought that the nature of the feedback provided a key role in its preference. For instance, when the feedback becomes more explanatory, it was considered a general comment all students and it becomes a model for similar codes. Cho and MacArthur (2010) also suggest providing this kind of feedback can result in positive outcomes.

Some of the students emphasized that using GitHub was positively contributed to the students' receiving feedback. In this sense, GitHub has some useful features for asynchronous communication, immediate feedback, transmitting content and it supports to evaluate students. When the instructor and students entered the GitHub at the same time, the amount of feedback increased in the current study. Because, the students rely on to be responded by the instructor, and they feel flexible to continue to the post to the instructor like a conversation on GitHub. This finding is in parallel with the idea of some researchers that many students prefer quick feedback (Rae & Cochrane, 2008). One can infer from the results that, when the feedback giving period is short, the students should be careful to keep in mind what the need was and how does the feedback fulfills the need.

Overall, the study concluded that the most frequently used feedback type was explanatory and corrective feedback. The Spearman Rho correlation proofs this foresight that the highest correlation between the feedback types and the programming performance scores was the explanatory, and corrective feedback. Even if the explanatory feedback was provided mostly in this study, it is difficult to match the feedback type and programming performances clearly. Because there are several factors influencing the feedback use such as the nature of projects,

the features of the GitHub environment and the role of instructor. The results indicated that various types of feedback provided positive effects on the online instructional process of programming. One more contribution of this study is that; online programming process sometimes requires combining various types of feedback.

This study has certain limitations that should be enumerated to create opportunities for future research. First, the study utilized a small size of students, the selection and size of the study subjects may limit the generalization of the results. For generalization, future studies to investigate feedback with a broader sample. The data were also interpreted through affordances of GitHub, that different environments for feedback interactions were warranted.

5. Implications for Future Research

In online programming instructions, providing useful feedback is somewhat challenging and time-consuming for the instructor. Considering the affordances of the online platform and the nature of feedback, cautions should be taken when deciding to provide feedback for certain types of programming tasks. Instructors should deal with the affordances of the online learning environment and also should have experiences about learners' needs. In this study, the effect of the feedback was considered through the programming performances on the tasks. A further study may explain relationships between the nature of the tasks, the type of skills or knowledge required to task completion and the nature of the feedback provided for the tasks.

Consequently, this study addressed using various types of feedback in online instruction via GitHub. The results are hoped to guide instructors and instructional designers who may wish to organize courses through the various use of online feedback.

References

Baker, W., & Hansen Bricker, R. (2010). The effects of direct and indirect speech acts on native English and ESL speakers' perception of teacher written feedback. *System*, *38(1),* 75–84.

Baldwin, L. P., & Kuljis, J. (2000). Visualisation techniques for learning and teaching programming. *Journal of Computing and Information Technology*, 8(4), 285–291.

Baran, E., Correia, A. P., & Thompson, A. (2011). Transforming online teaching practice: critical analysis of the literature on the roles and competencies of online teachers. *Distance Education, 32(3),* 421–439.

Barr, V., & Guzdial, M. (2015). Advice on teaching CS, and the learnability of programming languages. *Communications of the ACM*, *58(3),* 8–9.

Benachour, P., & Edwards, R. (2009). Animation and interactive programming: A practical approach. *Electronic Notes in Theoretical Computer Science, 224,* 133–140.

Campos, D. S., Mendes, A. J., Marcelino, M. J., Ferreira, D. J., & Alves, L. M. (2012). *A multinational case study on using diverse feedback types applied to introductory programming learning.* Frontiers in Education Conference proceedings (FIE 2012) (pp. 594–599). Seattle, Washington.

Cho, K., & MacArthur, C. (2010). Student revision with peer and expert reviewing. *Learning and Instruction*, 20(4), 328–338.

Duffy, T. M., & Kirkley, J. R. (2004). *Learning theory and pedagogy applied in distance learning: The case of Cardean University.* Learner-Centered Theory and Practice in Distance Education: Cases from Higher Education, 107–141.

Eng, T. H., Ibrahim, A. F., & Shamsuddin, N. E. (2015). Students' perception: Student Feedback Online (SuFO) in higher education. *Procedia - Social and Behavioral Sciences*, *167,* 109–116.

Ferris, D. R. (1995). Student reactions to teacher response in multiple-draft composition classrooms. *TESOL*

Quarterly, *29(1),* 33–53.

Fini, A. (2009). The technological dimension of a massive open online course: The case of the CCK08 course tools. *The International Review of Research in Open and Distributed Learning*, *10(5).*

Gikandi, J. W., & Morrow, D. (2016). Designing and implementing peer formative feedback within online learning environments. *Technology, Pedagogy and Education*, *25(2),* 153–170.

Gikandi, J. W., Morrow, D., & Davis, N. E. (2011). Online formative assessment in higher education: A review of the literature. *Computers & Education*, *57(4)*, 2333–2351.

Gunnarsson, S., Larsson, P., Månsson, S., Mårtensson, E., & Sönnerup, J. (2017). *Enhancing student engagement using GitHub as an educational tool.* Lund University. Retrieved from https://www.lunduniversity.lu.se/lup/publication/

Hatziapostolou, T., & Paraskakis, I. (2010). Enhancing the impact of formative feedback on student learning through an online feedback system. *Electronic Journal of e-Learning*, *8(2),* 111-122.

Hattie, J., & Timperley, H. (2007). The power of feedback. *Review of Educational Research*, 77(1), 81–112.

Heo, M., & Chow, A. (2005). The impact of computer augmented online learning and assessment tool. *Educational Technology & Society*, *8,* 113–125.

Huxham, M. (2007). Fast and effective feedback: are model answers the answer? *Assessment & Evaluation in Higher Education*, *32(6),* 601–611.

Hyland, F. (2001). Providing effective support: Investigating feedback to distance language learners. *Open Learning: The Journal of Open, Distance and e-Learning*, *16(3),* 233–247.

Hyland, F. (2010). Future directions in feedback on second language writing: Overview and research agenda. *International Journal of English Studies*, *10(2),* 171–182.

Jara, M., & Mellar, H. (2010). Quality enhancement for e-learning courses: The role of student feedback. *Computers & Education*, *54(3),* 709-714.

Jonsson, A. (2013). Facilitating productive use of feedback in higher education. *Active Learning in Higher Education*, *14(1),* 63–76.

Kluger, A. N., & DeNisi, A. (1996). The effects of feedback interventions on performance: A historical review, a meta-analysis, and a preliminary feedback intervention theory. *Psychological Bulletin*, *119(2),* 254–284.

Kop, R. (2011). The challenges to connectivist learning on open online networks: Learning experiences during a massive open online course. *The International Review of Research in Open and Distributed Learning*, *12(3),* 19–38.

Kyrilov, A. (2017). *Using case-based reasoning to improve the quality of feedback provided by automated assessment systems for programming exercises.* (Unpublished doctoral dissertation). UC Merced.

Law, K. M. Y., Lee, V. C. S., & Yu, Y. T. (2010). Learning motivation in e-learning facilitated computer programming courses. *Computers & Education, 55(1),* 218–228.

Lawless, K. A., & Pellegrino, J. W. (2007). Professional development in integrating technology into teaching and learning: knowns, unknowns, and ways to pursue better questions and answers. *Review of Educational Research*, *77(4),* 575–614.

Lipnevich, A. A., & Smith, J. K. (2009). Effects of differential feedback on students' examination performance. *Journal of Experimental Psychology: Applied*, *15(4),* 319–333.

Ludwig-Hardman, S., & Dunlap, J. C. (2003). Learner Support Services for Online Students: Scaffolding for success. *International Review of Research in Open and Distance Learning*, *4(1).*

Mishra, P., & Koehler, M. J. (2006). Technological pedagogical content knowledge: A framework for teacher knowledge. *Teachers College Record*, *108(6)*, 1017–1054.

McGill, T. J., & Volet, S. E. (1997). A conceptual framework for analyzing students' knowledge of programming. *Journal of research on Computing in Education*, *29(3),* 276-297.

Naidu, S. (2007). *Instructional design for optimal learning. In G. M. Moore (Ed.), Handbook of distance education*. Mahwah, NJ: Lawrence Erlbaum Associates. 247–258

Nicol, D. J., & Macfarlane-Dick, D. (2006). Formative assessment and self-regulated learning: A model and seven principles of good feedback practice. *Studies in Higher Education*, *31(2),* 199-218.

Niessen, A. S. M., Meijer, R. R., & Tendeiro, J. N. (2016). Predicting performance in higher education using proximal predictors. *PLoS ONE*, 11(4).

Orsmond, P., & Merry, S. (2011). Feedback alignment: effective and ineffective links between tutors' and students' understanding of coursework feedback. *Assessment & Evaluation in Higher Education*, *36(2),* 125–136.

Pitts, S. E. (2005). 'Testing, testing...': How do students use written feedback? *Active Learning in Higher Education*, *6(3),* 218–229.

Pyke, J. G., & Sherlock, J. J. (2010). A closer look at instructor-student feedback online: A case study analysis of the types and frequency. *Journal of Online Learning and Teaching*, *6(1),* 110-121.

Rae, A. M., & Cochrane, D. K. (2008). Listening to students: How to make written assessment feedback useful. *Active Learning in Higher Education*, *9(3),* 217–230.

Sáez-López, J. M., Román-González, M., & Vázquez-Cano, E. (2016). Visual programming languages integrated across the curriculum in elementary school: A two year case study using "Scratch" in five schools. *Computers & Education*, *97,* 129–141.

Schimmel, B. J. (1983). *A meta-Analysis of feedback to learners in computerized and programmed instruction.* Paper presented at the AREA, Montreal.

Shute, V. J. (2008). Focus on formative feedback. *Review of Educational Research*, *78(1),* 153-189.

Singh, R., Gulwani, S., & Solar-Lezama, A. (2013). Automated feedback generation for introductory programming assignments. *ACM SIGPLAN Notices*, *48(6),* 15-26.

Van Horne, S., Curran, M., Smith, A., VanBuren, J., Zahrieh, D., Larsen, R., & Miller, R. (2018). Facilitating student success in introductory chemistry with feedback in an online platform. *Technology, Knowledge and Learning*, *23(1),* 21–40.

Vasilyeva, E., Puuronen, S., Pechenizkiy, M., & Rasanen, P. (2007). Feedback adaptation in web-based learning systems. *International Journal of Continuing Engineering Education and Life Long Learning*, *17(4–5),* 337–357.

Vujoˇsevic-Janiˇcic, M., & Toˇsic, D. (2008). The role of programming paradigms in the first programming courses. *The Teaching of Mathematics*, *11(2),* 63–83.

Walker, M. (2009). An investigation into written comments on assignments: Do students find them usable?. *Assessment & Evaluation in Higher Education*, *34(1),* 67–78.

Weaver, M. R. (2006). Do students value feedback? Student perceptions of tutors' written responses. *Assessment & Evaluation in Higher Education*, *31(3),* 379-394.

Xia, B. S. (2015). Benefit and cost analysis of massive open online courses: Pedagogical implications on higher education. *International Journal of Cyber Behavior, Psychology and Learning (IJCBPL), 5(3),* 47-55.

Xia, B. S., & Liitiäinen, E. (2017). Student performance in computing education: an empirical analysis of online learning in programming education environments. *European Journal of Engineering Education*, *42(6),* 1025–1037.

Zaini, A. (2018). Word processors as monarchs: Computer-generated feedback can exercise power over and influence EAL learners' identity representations. *Computers & Education*, *120,* 112–126.

The Effect of Algorithm Education on Students' Computer Programming Self-Efficacy Perceptions and Computational Thinking Skills

Pınar Mıhcı Türker[1]
Ferhat Kadir Pala[1]

[1]Aksaray University, Faculty of Education

DOI: 10.21585/ijcses.v3i3.69

Abstract

In this study, the effect of algorithm education on pre-service teachers' computational thinking skills and computer programming self-efficacy perceptions were examined. In the study, one group pretest posttest experimental design was employed. The participants consisted of 24 (14 males and 10 females) pre-service teachers, majoring in Computer Education and Instructional Technology (CEIT). In order to determine the pre-service teachers' computer programming self-efficacy perceptions, the Computer Programming Self-Efficacy Scale was used, whereas Computational Thinking Skills Scale was used to determine their computational thinking skills. The Wilcoxon Signed-Rank Test was used to analyze the differences between pretest and posttest scores of students' computer programming self-efficacy perceptions and computational thinking skills. Throughout the practices, 10 different algorithmic problems were presented to the students each week, and they were asked to solve these problems using flow chart. For 13 weeks, 130 different algorithmic problems were solved. Algorithm education positively and significantly increased students' simple programming tasks, complex programming tasks and programming self-efficacy perceptions. On the other hand, algorithm education had a positive and significant effect only on students' algorithmic thinking sub-dimension but did not have any effect on other sub-dimensions and computational thinking skills in general.

Keywords: algorithm education, computational thinking, computer programming self-efficacy, pre-service teachers, computer education and instructional technology

1. Introduction

The process of creating software that brings technological equipment to life, in other words, the programming process, has gained importance today and has emerged as an important skill that every individual should have. Teaching processes have been shaped in this framework, and it has been ensured that programming education takes place as a course in schools starting from elementary school (Sayın, 2017; International Society for Technology in Education [ISTE], 2018). In Turkey, coding education has been included in Information Technologies and Software course starting from the 5th grade (11 years old) since 2012. Recently, it has come to the agenda to further reduce the age range of coding education and to include this content in all age groups from 1st grade to 4th grade (between 7-10 years old).

In short programming can be said as a series of codes written to make electronic devices perform certain functions in order to produce solutions to problems (Arabacıoğlu, Bülbül & Filiz, 2007; Blackwell, 2002; Ersoy, Madran & Gülbahar, 2011; Yükseltürk & Altıok, 2016). Programming skill is not only a skill required to produce software for computers. Students' high-level skills develop during the programming process. For example, Akçay and Çoklar (2016) describe these skills as critical thinking, algorithmic thinking, analytical thinking, problem solving, multidimensional thinking, creativity and questioning. There are also studies showing that programming skills have positive effects on computational thinking skills (Lye & Koh, 2014; Pala & Mıhcı Türker, 2019; Yıldız & Çiftçi, 2017). Computational thinking is a kind of analytical thinking, which includes elements such as problem solving, system design and understanding of human behavior based on the concepts of computer science (Wing, 2006). Korkmaz, Çakır & Özden (2017) gathered the computational thinking skills in five sub-dimensions on the scale they developed. These skills are creativity, algorithmic thinking, collaboration, critical thinking and problem solving. In this study, these sub-dimension of computational thinking skills were used.

The programming process takes place in the development stage of the software development process. The programming process in this stage consists of four stages. Algorithm development is at the second stage of these processes (Çamoğlu, 2018). Akçay and Çoklar (2016) states that the algorithm as the process of developing a design that reveals how the computer should work and the process of finding a solution to a problem. According to Gökoğlu (2017), the operations performed to solve the problem during the programming process vary according to the programming languages, but the logic does not change. To this end, students should be given the logic of programming before moving on to any programming language teaching. Algorithms form the basis of programming logic. In other words, without any programming language, students are taught the logic of programming with algorithm. With algorithm, it is possible for the individual to write the processing steps of the program in his or her own language, that is, to create the flowchart of the program with the so-called code. When the basic steps of algorithm development are examined, individuals should first identify the problem, define the inputs and outputs and determine possible solutions. Then, they should parse these solutions into steps and connect these steps to each other, that is, building an algorithm (Çamoğlu, 2018). In this process, it is possible that the students will benefit from some sub-dimensions of computational thinking skills such as algorithmic thinking, problem solving and creativity. However, many studies indicated that programming is a difficult process and there are some problems in programming education (Arabacıoğlu, Bülbül and Filiz, 2007; Gomes and Mendes, 2007; Esteves and Mendes, 2004; Hongwarrittorn and Krairit, 2010; Ozoran, Çağıltay and Topalli, 2012; Robins, Rountree and Rountree, 2003; Saygıner and Tüzün, 2017). The complex structure of the programming language is listed as one of these problems (Gomes and Mendes, 2007; Kalelioğlu, 2015). Aşkar and Davenport (2009) state that the programming course at the university is perceived as quite difficult by the beginner-level students. Students' initial acceptance of programming as difficult can cause them to fail in this course. For example, Altun and Mazman (2012) emphasize that students may fail due to the fact that students accept programming as difficult and have low self-efficacy perceptions.

In the light of this information, it can be said that with the algorithm education, the student will get the logic of programming and focus on the solution process of the problems. In other words, without the complex language of programming, it is provided to comprehend the logic of programming and enter the programming course. On the other hand, algorithm education is one of the computer science unplugged activities. With

computer science unplugged activities, it is aimed to teach computer science to students without computers. Bell (2014) states that with computer science unplugged activities, students interact directly with content and not bothered with unnecessary details. In addition, computer science unplugged enables them to learn how computing principles work, providing students with the opportunity of direct observation and experimentation. Computer science unplugged activities include many subjects such as algorithms, human computer interaction, artificial intelligence, computer graphics, data compression and encryption (Bell, 2014). Bell, Alexander, Freeman, and Grimley (2009) state that engaging in activities away from the computer might be effective for students because they might see the computer as a tool or toy. Students may think of problems encountered as a computer programmer by getting away from the computer. They can learn subjects such as algorithms, data compression, graphics algorithms, interface design, and models of computing without technical experience. In most cases, students find programming topics impressive. For example, in a study conducted by Mıhcı Türker and Pala (2018) with 5th and 6th graders, students stated that coding helped with developing games, moving characters, making movies, making robots and having fun. However, although the students have positive views on programming, the difficulty of learning programming should be explained appropriately. Therefore, computer science unplugged provides students with the opportunity to interact with the topic they find impressive without getting into complex programming topics, provides more fun and lasting learning opportunities in different contexts such as game-based learning and learning by discovery and help them attain computational thinking skills (Kalelioglu, 2017). Kalelioğlu (2015) addressed the difficulties of programming languages during the programming process and stated that thinking skills should be supported with different methods for programming. In this context, computer science unplugged is important for developing these skills.

In line with these views, it is possible that students will learn the logic of programming and develop their programming self-efficacy through algorithm education by moving away from the complex structure of programming languages. On the other hand, students are expected to produce a solution to a problem without programming language during this process and to use high-level thinking skills in this way. Therefore, in this study, the effect of algorithm education on pre-service teachers' computational thinking skills and computer programming self-efficacy perceptions was examined and answers to the following questions were investigated:

1) Is there a significant difference between the pretest and posttest scores of students' computer programming self-efficacy perceptions?

2) Is there a significant difference between the pretest and posttest scores of students' computational thinking skills?

1.2 Related Literature

In this study, the effect of algorithm education on pre-service teachers' computer programming self-efficacy perceptions and computational thinking skills were examined. For example, in a study conducted by Tsai (2019), visual programming language was used to teach basic programming concepts to university students. Accordingly, students' learning performances and computer programming self-efficacy regarding perceptions were examined. After the implementation, students' performances in basic programming concepts and programming self-efficacy increased positively.

Özmen and Altun (2014) conducted a study on students' perceptions of self-efficacy regarding programming.

In this study, the reasons of failure of teacher candidates in programming course were examined and 12 students were interviewed for this purpose. The interview results revealed that students had difficulties in processes such as programming knowledge, programming skills, understanding the logic of the program and debugging. Students stated that the main reasons for their failure in programming were lack of practice and knowledge and not being able to develop algorithms. On the other hand, it is seen that students with high programming experience have high programming achievement and high self-efficacy perceptions.

In another study conducted by Mazman and Altun (2013), self-efficacy perceptions and pre-experience characteristics of Computer Education and Instructional Technology Education (CEIT) students were examined. In this context, programming courses were given to the students who had taken programming courses before, scales were applied before and after the course and the obtained data were analyzed. Accordingly, the programming course significantly increased computer programming self-efficacy perceptions in both pre-experienced and experienced groups, and this increase was found to be higher in the group without prior experience. In addition, the difference between self-efficacy perceptions between pre-experienced and experienced groups decreased at the end of the programming course. The researchers stated that students' self-efficacy perceptions and their previous knowledge played an important role in their achievement in this course.

In a study conducted by Davidson, Larzon and Ljunggren, (2010), the effect of Introduction to Programming course on the change in self-efficacy perceptions of students was examined. Accordingly, individuals' self-efficacy scores did not show a significant change at the beginning and at the end of the programming course. However, there was an increase in students' sub-skills such as analysis and solution of simple problems, debugging and working principle of computer.

In the light of the findings obtained in the studies, it is concluded that the students' perceptions of self-efficacy increased with the increase of their experience in programming and consequently their success in programming increased. Based on these data, it is possible to obtain programming logic and to increase self-efficacy perceptions positively without facing problems related to language or abstract structure in programming.

Pala and Mıhcı Türker (2019) examined the effect of programming education on the computational thinking skills of pre-service teachers. In this context, the researchers found that robotic-based programming significantly affected students' creativity, algorithmic thinking and critical thinking dimensions and that text-based programming had no effect. In a different study, Oluk and Korkmaz (2016) determined a positive relationship between programming skills and computational thinking skills. As students' programming skills increased, their computational thinking skills also increased. Similarly, as a result of implementations done using programming, Fadjo (2012) determined that implementations play an important role in the development of students' computational thinking skills and concept knowledge.

In addition, there are studies showing the effectiveness of programming education on sub-skills comprising the computational thinking skills such as problem solving (Kalelioğlu & Gülbahar, 2014; Kukul & Gökçearslan, 2014; Somyürek, 2014; Robinson, 2005), creativity (Kobsiripat, 2015).

2. Method

Examining the effect of algorithm education on pre-service teachers' computer programming self-efficacy perceptions and computational thinking skills, the present study employed one group pretest posttest experimental design, one of the quasi-experimental research designs. In this design, the significance of the difference between the pretest and posttest values of the groups is tested (Cohen, Manion & Morrison, 2002).

In the study, unplugged implementations in the algorithm course made up the independent variable, whereas students' computer programming self-efficacy perception and computational thinking skill made up the dependent variable.

2.1 Study Group

The study was conducted within the scope of Algorithm course during the Spring semester of 2018-2019 academic year. The participants consisted of 24 (14 males and 10 females) pre-service teachers studying at the CEIT department of a university in Central Anatolia Region, Turkey. The age range of the participants ranged from 19 to 26, and all of them had programming knowledge.

2.2 Data Collection Tool

In order to determine the pre-service teachers' computer programming self-efficacy perceptions, the Computer Programming Self-Efficacy Scale, adapted to Turkish by Altun and Mazman (2012), was used. Computational Thinking Skills Scale (CTSS), developed by Korkmaz, Çakır and Özden (2017) for undergraduate students, was used to determine pre-service teachers' computational thinking skills.

The original computer programming self-efficacy scale consists of 32 items and four factors. However, after the adaptation and analysis by Altun and Mazman (2012), the Turkish form consists of nine items and two factors, simple programming tasks and complex programming tasks. Internal consistency for the scale was .90 for the first factor, .94 for the second factor, and .92 for the overall scale, and these nine items explained 80.81% of the total variance.

Computational Thinking Skills Scale (CTSS), developed by Korkmaz, Çakır and Özden (2017) for undergraduate students, was used to determine pre-service teachers' computational thinking skills. The scale has 29 items and five dimensions, creativity (eight items), algorithmic thinking (six items), collaboration (four items), critical thinking (five items) and problem solving (six items). The scale is a five-point Likert one. Internal consistency coefficient Cronbach's alpha values were .84 for the creativity dimension, .86 for the algorithmic thinking dimension, .86 for the collaboration dimension; .78 for the critical thinking dimension, .72 for the problem-solving dimension and .82 for the overall scale. The explained variance for all factors was 56.1%.

2.3 Data Analysis

Russell and Purcell (2009) stated that parametric tests should not be used with groups less than 30, and that when the size of the group is smaller ($n<30$), the data do not meet normality. Gosling (1995) stated that when the distribution of the universe is unknown and when the group number is small ($n<30$), the normality cannot be adequately met, and suggested the use of non-parametric tests. Ploger and Yasukawa (2003) suggested that parametric techniques should be used when the groups are large ($n> 30$) and the normality is met. Since the number of groups included in the study was less than 30, non-parametric tests were used to compare between pretest and posttest scores of students. In this respect, the Wilcoxon Signed-Rank Test was used to analyze the differences between pretest and posttest scores of students' computer programming self-efficacy perceptions and computational thinking skills.

2.4 Implementation Process

The implementation process of the study was conducted with 24 pre-service teachers who were studying in the CEIT department of a university in Central Anatolia during the 2018-2019 academic year. First, scales

were administered to the pre-service teachers, and then the process was continued by presenting various algorithm problems for 13 weeks. Each week, 10 different algorithmic problems were presented to the students, and they were asked to solve these problems using flowchart. A total of 130 different algorithmic problems were solved.

The problems were prepared by researchers using three different algorithm books (Çamoğlu, 2018; Tungut, 2016; Vatansever, 2015). The problems were approved by two experts. According to this, questions about sequential operations, conditional states and repetitive structures are included in the algorithm problems. The students exchanged information with their groupmates during the solution process and discussed the solution with them. At the end of the process, the scales were applied to the participants again, and the data obtained were organized for analysis.

3. Results

3.1 Is there a significant difference between the pretest and posttest scores of students' computer programming self-efficacy perceptions after the algorithm education?

Within the scope of the study, in order to determine pre-service teachers' computer programming self-efficacy perceptions, Computer Programming Self-Efficacy Scale was administered. The scale was comprised of two dimensions and nine items. Descriptive analyzes were conducted in accordance with the responses given by the pre-service teachers to the items, and the results are presented in Table 1.

Table 1. Descriptive statistics regarding pre-service teachers' programming self-efficacy perceptions

	Dimension	Item Number	*N*	Lowest	Highest	$\bar{X}$	*ss*
Before Implementation	Simple programming tasks	3	24	5	21	14.41	5.44
	Complex programming tasks	6	24	6	36	20.29	9.10
Total	Programming self-efficacy perceptions	9	24	11	57	34.70	13.76
After Implementation	Simple programming tasks	3	24	12	21	17.79	3.07
	Complex programming tasks	6	24	7	40	24.79	7.56
Total	Programming self-efficacy perceptions	9	24	21	61	42.58	9.99

According to Table 1, while pre-service teachers' mean scores of simple programming tasks were $\bar{X}$=14.41 before the implementation and $\bar{X}$=17.79 after the implementation, pre-service teachers' mean scores of complex programming tasks were $\bar{X}$ =20.29 before the implementation and $\bar{X}$ =24.79 after the implementation. In addition, pre-service teachers' computer programming self-efficacy perceptions were 11 at the lowest before the implementation and 21 after the implementation, whereas pre-service teachers' computer programming self-efficacy perceptions were 57 at the highest 57 before the implementation and 61 at the highest after the implementation.

Table 2. Pre-service teachers' programming self-efficacy perception pretest posttest score analysis

Dimension	Test	Ranks	N	Mean	Total	*Z*	*p*
Simple Programming Tasks	Post Pre	Negative Ranks	4			-3.30	.001*
		Positive Ranks	16	4.25	17		
		Equal	4	12.06	193		
		Total	24				
Complex Programming Tasks	Post Pre	Negative Ranks	8			-2.43	.015*
		Positive Ranks	16	8.13	65		
		Equal	0	14.69	235		
		Total	24				
Programming Self-Efficacy Perception	Post Pre	Negative Ranks	6			-3.03	.002*
		Positive Ranks	17	6.42	38.5		
		Equal	1	13.97	237.5		
		Total	24				

** $p < .05$*

Table 2 presents the results of Wilcoxon Signed-Rank Test done to determine whether pre-service teachers' computer programming self-efficacy pretest and posttest scores differ significantly. Accordingly, there is a significant difference in pre-service teachers' pretest and posttest scores in all dimensions. However, since positive ranks were taken as baseline and the z value was negative, the difference was in favor of the posttest (p<.05; z_{simple}: -3.30; $z_{complex}$:-2.43; z_{total}: -3.03). In other words, algorithm education significantly increases pre-service teachers' computer programming self-efficacy perceptions.

3.2 Is there a significant difference between the pretest and posttest scores of students' computational thinking skills after the algorithm education?

Within the scope of the study, in order to determine pre-service teachers' computational thinking skills, CTSS was administered. The scale was comprised of five dimensions and 29 items. Descriptive analyzes were conducted in accordance with the responses given by the pre-service teachers to the items, and the results are presented in Table 3.

Table 3. Descriptive statistics of students' computational thinking skills

	Dimension	Item Number	N	Lowest	Highest	$\bar{X}$	ss
Before Implementation	Creativity	8	23	24	40	34.21	3.84
	Algorithmic Thinking	6	23	6	29	17.26	6.94
	Collaboration	4	23	6	30	13.82	5.95
	Critical Thinking	5	23	4	20	14.69	3.92
	Problem Solving	6	23	12	25	18.00	3.60
Total	Computational Thinking	29	23	82	142	98.00	14.31

After Implementation	Creativity	8	23	26	40	34.86	3.55
	Algorithmic Thinking	6	23	6	30	20.34	7.13
	Collaboration	4	23	8	27	12.56	5.22
	Critical Thinking	5	23	13	20	14.17	3.66
	Problem Solving	6	23	6	25	19.26	4.01
Total	Computational Thinking	29	23	79	125	101.21	12.92

Table 3 shows that one pre-service teacher's scale results were considered invalid, and the analysis was continued with 23 pre-service teachers. According to this, pre-service teachers' mean scores of computational thinking skills were $\bar{X}$=98.00 before the implementation and $\bar{X}$=101.21 after the implementation. In addition, pre-service teachers' computational thinking skills were 82 at the lowest before the implementation and 79 after the implementation, whereas pre-service teachers' computational thinking skills were 142 at the highest 57 before the implementation and 125 at the highest after the implementation.

Table 4. Pre-service teachers' computational thinking skills pretest posttest score analysis

Dimension	Test	Ranks	N	Mean	Total	*Z*	*P*
Creativity	Post Pre	Negative Ranks Positive Ranks Equal Total	6 11 6 23	9.83 8.55	59.00 94.00	-.831	.406
Algorithmic	Post Pre	Negative Ranks Positive Ranks Equal Total	3 17 3 23	11.50 10.32	34.50 175.50	-2.64	.008*
Collaboration	Post Pre	Negative Ranks Positive Ranks Equal Total	14 5 4 23	9.43 11.60	132.00 58.00	-1.49	.135
Critical Thinking	Post Pre	Negative Ranks Positive Ranks Equal Total	9 10 4 23	11.50 8.65	103.50 86.50	-.343	.732
Problem Solving	Post Pre	Negative Ranks Positive Ranks Equal Total	4 15 4 23	12.75 9.27	51.00 139.00	-1.77	.075
Total	Post Pre	Negative Ranks Positive Ranks Equal	8 14 1	9.88 12.43	79.00 174.00	-1.54	.123

	Total	23

** $p < .05$*

Table 4 presents the results of Wilcoxon Signed-Rank Test done to determine whether pre-service teachers' computational thinking skills pretest and posttest scores differ significantly. Accordingly, there is a significant difference in pre-service teachers' algorithmic thinking pretest and posttest scores. Since positive ranks were taken as baseline and the z value was negative, the difference was in favor of the posttest (p<.05; $z_{algorithmic}$: -2,64). However, there was no significant difference in pre-service teachers' computational thinking skills and sub-dimension comprising this skill (p>.05).

4. Discussion

In this study, the effect of algorithm education on pre-service teachers' computational thinking skills and self-efficacy perceptions related computer programming were investigated. In this context, a 13-week implementation process was carried out with 24 pre-service teachers (14 males, 10 females). Each week, 10 different algorithmic problems were presented to the students and they were asked to solve these problems using flow chart. A total of 130 different algorithmic problems were solved. The students exchanged information with their groupmates during the solution process and discussed the solution with them.

In this study, the effect of direct programming logic on pre-service teachers' computational thinking skills and computer programming self-efficacy perceptions were examined without any programming language because many programming languages cause negative attitudes in students with their abstract and complex structure (Gomes & Mendes, 2007; Kalelioğlu, 2015).

In this direction, the study results show that algorithm education positively and significantly increases students' perceptions of simple programming tasks, complex programming tasks and programming self-efficacy. Similarly, related literature put forth that students' computer programming self-efficacy perceptions are positively affected as a result of adequate understanding of programming logic by students (Jegede, 2009; Mazman & Altun, 2013; Özmen & Altun, 2014; Tsai, 2019). On the other hand, it is possible to come across studies that do not show a significant change in students' self-efficacy after programming education (Davidson, Larzon & Ljunggren, 2010).

Bell, Alexander, Freeman & Grimley, (2009) stated that students do not deal with the difficulty of programming and mobilize their computational thinking skills through computer science unplugged. Similarly, Kalelioğlu (2015) pointed out the importance of computer science unplugged in the development of high-level thinking skills. Within the scope of the present study, algorithm education was offered using computer science unplugged. At the end of the implementation process, it was found that the algorithm education positively and significantly affected the pre-service teachers' algorithmic thinking skills. However, it had no effect on other sub-skills and computational thinking skill in general. Significant differences in algorithmic thinking are expected. On the other hand, it is surprising that there was no significant difference in other sub-dimensions and overall score. It was expected that an increase in algorithmic thinking skill would affect creativity, collaboration, critical thinking, problem solving and computational thinking skill in general. Yet, no significant difference was found because during the algorithm education, students were allowed to work together in a collaborative environment, and they were able to find solutions to problems together. During the process, students discussed the solutions with a critical eye, and they created new solutions. This

is believed to be due to the abstract progress of the implementations. It is thought that more clear data can be obtained by comparing the implementations in which the students can be more active in different researches and the implementations in which the process of solving algorithmic problems can be carried out passively. The studies examining the relationship between programming education and computational thinking skills show that robotic programming education (Grover & Pea, 2013; Lee et al., 2011; Pala & Mıhcı Türker, 2019; Penmetcha, 2012; Repenning, Webb and Ioannidou, 2010) and visual-based programming education (Fadjo, 2012; Howland & Good, 2015) are effective on students' computational thinking skills. In addition, both robotic and visual-based programs (Khanlari, 2013; Kobsiripat, 2015) have positive effects on algorithmic thinking sub-dimension (Penmetcha, 2012), critical thinking skill sub-dimension (Blanchard, Freiman & Lirrete-Pitre, 2010; Chambers, Carbonaro, Rex & Grove, 2007), collaboration sub-dimension (Khanlari, 2013; Miller & Nourbakhsh; 2016) and problem solving skill dimension (Atmatzidou & Demetriadis, 2012; Khanlari, 2013; Petre & Price, 2004; Robinson, 2005; Rogers & Portsmore, 2004). This is believed to be due to the fact that these platforms have a more concrete structure and are easier to understand programming logic. In future studies, it is possible to compare all three methods of implementation by using different groups. In addition, this study was conducted with a limited sample because of the size of class. However, it is assumed that other latent variables such as projects in other courses taken during the study period do not affect dependent variables. Similar implementations can be carried out with larger experiment groups and control groups.

In addition, in different studies, algorithm development through digital applications can be investigated. Thus, the effect of digital applications on pre-service teachers' self-efficacy and computational thinking skills can be examined. Similar studies can be reported in different age groups. Then, the success of these students towards programming can be examined.

5. Conclusion

It is possible to increase the self-efficacy of programming positively by learning the logic of programming by moving away from the complex structure of programming languages. In this respect, algorithms that are included in computer science unplugged activities and aim to give students the logic of programming without complex language structure gain importance. The student is required to identify the problem, to define the inputs and the outputs, and to determine the possible solutions during the algorithm creation stage. The next step is to parse these solutions into steps and associate these steps with each other. In this process, it is possible to utilize and develop various high-level thinking skills. Computational thinking is among these high-level thinking skills. In the light of this process, the effect of algorithm education, which is one of the computer science unplugged activities, on students' self-efficacy perception and computational thinking skills was examined in this study.

The results show that; algorithm education increases students' self-efficacy perceptions related programming in a positive and meaningful way. In this way, after the algorithm education, self-efficacy related programming can be increased, and students can start programming with more positive perspectives. On the other hand, there was no significant difference in the students ' computational thinking skills. However, only the algorithmic thinking, a sub-dimension of computational thinking, increased significantly and positively.

References

Akçay, A. & Çoklar, A. N. (2016). Bilişsel becerilerin gelişimine yönelik bir öneri: Programlama eğitimi. In A. İşman, H. F. Odabaşı & B. Akkoyunlu (Eds.), *Eğitim Teknolojieri Okumaları 2016* (pp. 121-139). Ankara, Turkey: The Turkish Online Journal of Educational Technology (TOJET).

Altun, A., & Mazman, S. G. (2012). Programlamaya ilişkin öz yeterlilik algısı ölçeğinin Türkçe formunun güvenirlik ve geçerlik çalışması, *Eğitimde ve Psikolojide Ölçme ve Değerlendirme Dergisi*, *3*(2), 297-308.

Arabacıoğlu, C., Bülbül, H. & Filiz, A. (2007, January). *A new approach to computer programming teaching*. Presented at the 2007 Academic Informatics Conference, Dumlupinar University, Kütahya, Turkey. Retrieved from https://ab.org.tr/ab07/kitap/arabacioglu_bulbul_AB07.pdf.

Aşkar, P., & Davenport, D. (2009). An investigation of factors related to self-efficacy for Java Programming among engineering students. *Online Submission*, *8*(1), 26-32.

Atmatzidou, S., & Demetriadis, S. N. (2012, July). Evaluating the role of collaboration scripts as group guiding tools in activities of educational robotics: Conclusions from three case studies. In *2012 IEEE 12th International Conference on Advanced Learning Technologies* (pp. 298-302). IEEE.

Bell, T. (2014). Ubiquity Symposium: The science in computer science: unplugging computer science to find the science. *Ubiquity,* March*,* 1-6. http://dx.doi.org/10.1145/2590528.2590531.

Bell, T., Alexander, J., Freeman, I., & Grimley, M. (2009). Computer science unplugged: School students doing real computing without computers. *The New Zealand Journal of Applied Computing and Information Technology*, 13(1), 20-29.

Blackwell, A. F. (2002, June). What is programming? Proceedings of the *14th workshop of the Psychology of Programming Interest Group,* Brunel University, Middlesex, UK. pp. 204-218.

Blanchard, S., Freiman, V., & Lirrete-Pitre, N. (2010). Strategies used by elementary schoolchildren solving robotics-based complex tasks: Innovative potential of technology. *Procedia-Social and Behavioral Sciences*, *2*(2), 2851-2857.

Chambers, J. M., Carbonaro, M., Rex, M., & Grove, S. (2007). Scaffolding knowledge construction through robotic technology: A middle school case study. *Electronic Journal for the Integration of Technology in Education*, 6, 55-70.

Cohen, L., Manion, L., & Morrison, K. (2002). *Research methods in education (6th ed.).* Abingdon, UK: Routledge

Çamoğlu, K. (2018). *Algorithm (6th ed.).* Kodlab: İstanbul, Turkey.

Davidson, K., Larzon, L. & Ljunggren, K. (2010). *Self-Efficacy in programming among STS students.* Retrieved from http://www.it.uu.se/edu/course/homepage/datadidaktik/ht10/reports.

Ersoy, H., Madran, R. O., & Gülbahar, Y. (2011, February). *A model proposed for teaching programming languages: Robotic programming*. Presented at the 2011 Academic Informatics Conference, İnönü University, Malatya, Turkey. Retrieved from https://ab.org.tr/ab11/kitap/ersoy_madran_AB11.pdf.

Esteves, M., & Mendes, A. J. (2004, October). A simulation tool to help learning of object oriented programming basics. In *34th Annual Frontiers in Education, 2004. FIE 2004*. (pp. F4C-7). IEEE.

Fadjo, C. L. (2012). *Developing computational thinking through grounded embodied cognition* (Unpublished dissertation). Columbia University, New York, NY, USA.

Gomes, A. & Mendes, A. J. (2007, September). Learning to program - difficulties and solutions. In *Proceedings of the International Conference on Engineering Education*. Coimbra, Portugal. Retrieved from http://icee2007.dei.uc.pt/proceedings/papers/411.pdf.

Gosling, J. (1995). *Introductory statistics*. Leichhardt, Australia: Pascal Press.

Gökoğlu, S. (2017). Algorithm perception in programming education: A metaphor analysis. *Cumhuriyet International Journal of Education,* 6(1), 1-14.

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

Hongwarrittorn, N., & Krairit, D. (2010, July). *Effects of program visualization (jeliot3) on students' performance and attitudes towards java programming.* Presented at the 8th International Conference on Computing, Communication and Control Technologies, Delhi, India.

Howland, K., & Good, J. (2015). Learning to communicate computationally with Flip: A bi-modal programming language for game creation. *Computers & Education*, *80* (2015), 224-240.

International Society for Technology in Education- ISTE (2018). *Computational thinking for all.* Retrieved from https:// www.iste.org/explore/articleDetail?articleid=152.

Jegede, P. O. (2009). Predictors of java programming self-efficacy among engineering students in a Nigerian University. *International Journal of Computer Science and Information Security,4*(1&2). Retrieved from https://arxiv.org/ftp/arxiv/papers/0909/0909.0074.pdf.

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.

Kalelioğlu, F. (2015). A new way of teaching programming skills to K-12 students: Code.org. *Computers in Human Behavior*, 52, 200-210.

Kalelioğlu, F. (2017). Bilgisayarsız bilgisayar bilimi (B3) öğretimi. Gülbahar, Y. (Ed.) *Bilgi İşlemsel Düşünmeden Programlamaya.* (I. Baskı) (ss. 183-206) Ankara, Turkey: Pegem Akademi.

Khanlari, A. (2013). Effects of robotics on 21st century skills. *European Scientific Journal (ESJ)*, 9 (27). 26-

36.

Kobsiripat, W., (2015). Effects of the media to promote the Scratch programming capabilities creativity of elementary school students. *Procedia-Social and Behavioral Sciences,* 174 (2015), 227-232.

Korkmaz, Ö., Çakir, R., & Özden, M. Y. (2017). A validity and reliability study of the Computational Thinking Scales (CTS). *Computers in Human Behavior*, 72, 558-569.

Kukul, V., & Gökçearslan, Ş. (2014, September). *Investigatıng the problem solving skills of students attended scratch programming course*. Presented at the 8th International Computer & Instructional Technologies Symposium, Trakya University, Edirne, Turkey.

Lee, I., Martin, F., Denner, J., Coulter, B., Allan, W., Erickson, J., Malyn-Smith, J. & Werner, L. (2011). Computational thinking for youth in practice. *Acm Inroads*, *2*(1), 32-37.

Lye, S. Y., & Koh, J. H. L. (2014). Review on teaching and learning of computational thinking through programming: What is next for K-12? *Computers in Human Behavior,* (41), 51-61.

Mazman, S. G., & Altun, A. (2013). The effect of introductory to programming course on programming self-efficacy of CEIT students. *Journal of Instructional Technologies and Teacher Education,* 2 (3), 24-29.

Mıhcı Türker, P. & Pala, F. K. (2018). Opinions of Secondary School Students, Teachers and Parents About Coding. *Elementary Education Online*, 17(4), 2013-2029.

Miller, D. P., & Nourbakhsh, I. (2016). Robotics for education. In *Springer handbook of robotics* (pp. 2115-2134). Springer, Cham.

Oluk, A., & Korkmaz, Ö. (2016). Comparing students' scratch skills with their computational thinking skills in terms of different variables. *I. J. Modern Education and Computer Science,* (11)*,* 1-7.

Ozoran, D., Çağıltay, N. E., & Topallı, D. (2012, November). Using scratch in introduction to programming course for engineering students. In *2nd International Engineering Education Conference (IEEC2012)* (Vol. 2, pp. 125-132).

Özmen, B., & Altun, A. (2014). Undergraduate students' experiences in programming: difficulties and obstacles. *Turkish Online Journal of Qualitative Inquiry*, *5*(3), 1-27.

Pala, F. K., & Mıhcı Türker, P. (2019). The effects of different programming trainings on the computational thinking skills. *Interactive Learning Environments*, 1-11. https://doi.org/10.1080/10494820.2019.1635495.

Penmetcha, M. R. (2012). *Exploring the effectiveness of robotics as a vehicle for computational thinking* (Doctoral dissertation), Purdue University, West Lafayette, IN, USA.

Petre, M., & Price, B. (2004). Using robotics to motivate 'back door'learning. *Education and Information Technologies*, 9 (2), 147-158.

Ploger, B. J., & Yasukawa, K. (2003). Introduction to statistics. In *Exploring Animal Behavior in Laboratory and Field* (pp. 415-446). San Diego, CA, USA: Academic Press.

Repenning, A., Webb, D., & Ioannidou, A. (2010, March). *Scalable game design and the development of a checklist for getting computational thinking into public schools.* Presented at the 41st ACM technical symposium on Computer science education, Milwaukee, Wisconsin, USA

Robins, A., Rountree, J., & Rountree, N. (2003). Learning and teaching programming: A review and discussion. *Computer science education*, 13(2), 137-172.

Robinson, M. (2005). Robotics-driven activities: Can they improve middle school science learning? *Bulletin of Science, Technology & Society*, 25 (1), 73-84.

Rogers, C., & Portsmore, M. (2004). Bringing engineering to elementary school. *Journal of STEM Education. Innovations & Research*, 5 (3), 17-28.

Russell, B., & Purcell, J. (2009). *Online research essentials: designing and implementing research studies* (Vol. 19). Hoboken, New Jersey, USA: John Wiley & Sons.

Saygıner, Ş. & Tüzün, H. (2017, December). *Difficulties in programming education and solutions.* In Proceedings of the International Computer and Instructional Technology Symposium, Malatya, Turkey. (pp 72-84).

Sayın, Z. (2017). Bilgisayar bilimi eğitimi kapsamı. Gülbahar, Y. (Ed.) *Bilgi İşlemsel Düşünmeden Programlamaya*. (I. Baskı) (ss. 133-154) Pegem Akademi, Ankara, Turkey.

Somyürek, S. (2015). An effective educational tool: construction kits for fun and meaningful learning. *International Journal of Technology and Design Education*, *25*(1), 25-41.

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.

Tungut, H. B. (2016). *Algorithm and Logic of Programming (6th ed.).* Kodlab: İstanbul, Turkey.

Vatansever, F. (2015). *Algorithm development and introduction to programming.* Seçkin: Ankara, Turkey.

Yıldız, M. & Çiftçi, E. (2017). Bilişimsel Düşünme ve Programlama. In H. F. Odabaşı, B. Akkoyunlu ve A. İşman (Eds.). *Eğitim Teknolojileri Okumaları 2017*, (Chapter 5, pp. 75-86). The Turkish Online Journal of Educational Technology (TOJET) and Sakarya University, Adapazarı, Turkey.

Yükseltürk, E. & Altıok, S. (2016). Pre-Service information technology teachers` perceptions about using Scratch tool in teaching programming. *Mersin University Journal of the Faculty of Education, 12*(1), 39-52.

IJCSES

Volume 3, No: 3

January 2020

ISSN 2513-8359