International Technology and Education Journal
Vol. 7 No. 2; December 2023
ISSN: 2602-2885
URL: http://itejournal.com/

Karalar, H. (2023). Adaptation of computer programming self-efficacy scale for computer literacy education into Turkish for middle scho ol students. International technology and education journal, 7(2), 51-59.

Adaptation of computer programming self-efficacy scale for computer literacy education into Turkish for middle school students

Halit Karalar, halit@mu.edu.tr, Mugla Sitki Kocman University, Turkey,

https://orcid.org/0000-0001-9344-9672

SUMMARY

The purpose of this study is to adapt the "Computer Programming Self-Efficacy Scale for Computer Literacy Education (CPSES)" developed by Tsai et al. (2019) into Turkish for middle school students and to develop a valid and reliable measurement tool. The participants of the study consisted of 348 eighth grade students. In order to test the factor structures of the adapted scale, the data obtained were analyzed with first and second order factor analysis. The results of the analysis showed that the adapted scale had high reliability, convergent validity and divergent validity. The results of the first- and second-order confirmatory factor analysis showed that both models had good fit values with the data. All 16 items and 5-factor structure (Logical Thinking, Cooperation, Algorithm, Control, Debug) in the original scale were retained in the Turkish scale. As a result, it was concluded that the Computer Programming Self-Efficacy Scale for Computer Literacy Education (CPSES) is a valid and reliable measurement tool in Turkish culture.

Keywords: Programming self-efficacy, middle school students, scale adaption, self-efficacy

INTRODUCTION

Recently, efforts to teach programming skills to K-12 students have become quite popular (Grover & Pea, 2013; Kafai & Burke, 2013; Resnick et al., 2009). Many countries are making changes in their curriculum to teach programming skills to students in elementary, middle, and even preschool levels (Falloon, 2016). So why do we teach programming to children? There is certainly no single answer to this question. The increasing demand for ICT professions necessary for technological and economic growth (Chen et al., 2017; Grover & Pea, 2013), the central role of programming in STEM and computational thinking (Brennan & Resnick, 2012; Lee, 2015; Lye & Koh, 2014; Wilson et al., 2010), its definition as a new skill within 21st-century skills and its contribution to the development of 21st-century skills (Clements & Gullo, 1984; Fessakis et al., 2013; Kafai & Burke, 2013), and its recognition as one of the literacy types of our age (Hagge, 2017) are some of the factors influencing this trend. Through programming, students are exposed to computational thinking, which involves the use of computer concepts such as abstraction, algorithmic thinking, pattern recognition and generalization, and iteration in problem-solving (Brennan & Resnick, 2012). In this context, equipping students with programming skills from an early age has become important, and programming instruction has been integrated into education systems worldwide.

How to teach programming to children in the concrete operational stage is an important question. The first study aimed at facilitating programming instruction in visual environments by concretizing the abstract concepts of programming is known as LOGO, developed for teaching mathematics to students (Papert, 1980). Studies show that learning programming through LOGO improves elementary students' mathematical reasoning and problem-solving skills (Papert, 1980) and enhances cognitive skills in preschool students (Clements, 1999). The pedagogy behind Papert's visual programming environment is constructionism. Based on constructivism, constructionism argues that learning occurs better when students are actively engaged in designing and creating a product (through the experiences gained during product development) instead of being passively presented with information (Papert, 1980). Papert also claims that students express their thoughts better and learn more when they build physical objects in a virtual environment.

In the 1990s, there was a decline in interest in programming education (Kafai & Burke, 2013; Moreno-León & Robles, 2016). Resnick (2012) attributed this decline in interest in programming during that period to the difficulty of learning the rules and syntax of traditional programming languages and the lack of engaging activities in classrooms. Nowadays, the interest in programming has been revived, especially among students at different developmental stages, through the use of block-based graphical programming environments (Grover & Pea, 2013). These user-friendly environments help make programming concepts more tangible, allowing students to see how commands work, test their ideas and receive immediate feedback, identify errors more easily, learn through trial and error, and engage in programming without dealing with the syntax of programming languages or memorizing command names (Kelleher & Pausch, 2005; Lye & Koh, 2014). These environments enable students to focus solely on design and creation (Grover et al., 2016; Grover & Pea, 2013). Examples of such environments based on LOGO are Alice and Scratch (Utting et al., 2010).

Research Article. Received: 07/10/2023 Checked with a plagiarism detection software

According to Resnick (2002), children who learn programming become technology authors by learning the language of technology and gain the ability to create with technology. Furthermore, as they learn programming, children also learn the design process: how to turn an idea into a project, how to experiment with new ideas, how to break down complex ideas into simpler parts, how to collaborate with others in a project, how to find and fix coding errors, and how to persevere when things are not working. While students may not pursue programming as a professional career, the design skills, creative thinking skills, and systematic thinking skills they acquire during this process will be useful regardless of the profession they choose.

Programming involves problem-solving and a production process that requires higher-order thinking skills (Saeli et al., 2011). When solving problems through programming, sequential operations are carried out systematically (Karalar, 2019): (1) Problem identification, (2) Breaking down the problem into manageable smaller parts, (3) Determining the sequential steps required to solve each part and expressing these steps visually (using algorithms or flowcharts) or verbally (using pseudocode), (4) Translating the visual or verbal sequential steps into a programming language that the computer can understand, (5) Identifying and correcting any coding errors.

According to Kazakoff & Bers (2014), in order to develop successful programs, students need to learn the logic of programming and possess systematic thinking skills. The first three stages described above, which involve understanding the logic of programming and problem-solving, are crucial for developing students' programming and problem-solving skills (Cooper et al., 2000). As a result of these stages, which also foster algorithmic thinking and problem-solving skills (Denner et al., 2012), students gain the ability to write programs easily in any programming language and can transfer their problem-solving skills to other domains (Palumbo, 1990). The fourth stage involves coding, which converts the sequentially identified steps for problem-solving into instructions that the computer can understand. As highlighted by Manches & Plowman (2015), programming and coding are distinct from each other; programming is the problem-solving process, while coding is the process of translating algorithms into instructions that the computer can understand. In the fifth stage, any coding errors are identified and corrected.

Several factors act as obstacles to successful programming learning for students. One common perception among students is that programming is difficult and uninteresting (Aktunc, 2013; Aşkar & Davenport, 2009), which adds to the challenges faced in programming instruction. Inadequate teaching methods employed by teachers (Guzdial & Soloway, 2002; Lee, 2011), the need for higher-order thinking skills in programming (Dann et al., 2000), familiarity with text-based programming languages and their syntax and commands (Jenkins, 2002), the abstract nature of fundamental programming concepts, and the challenges associated with testing and debugging programs (Pausch et al., 2000) are some of these challenges. Negative attitudes towards programming (Başer, 2013; Courte, Howard, & Bishop-Clark, 2006; Çetin & Özden, 2015; Korkmaz & Altun, 2013) and low self-efficacy beliefs (Altun & Mazman, 2012; Aşkar & Davenport, 2009; Tsai et al., 2019) are also factors contributing to these obstacles. Among these challenges, low programming self-efficacy is particularly significant in hindering students from successfully learning programming (Günbatar & Karalar, 2018; Tsai et al., 2019).

Self-efficacy, or self-efficacy beliefs, is a concept that emerged from Bandura's Social Learning Theory. It refers to an individual's judgment of their ability to organize and execute the necessary actions to perform a task, accomplish a goal, or produce a desired outcome (Bandura, 1997). Self-efficacy beliefs play a crucial motivational role in cognitive processes (Erol & Avcı Temizer, 2016). Bandura (1997) emphasizes that self-efficacy beliefs are one of the fundamental determinants of human behavior and behavioral changes, influencing not only actions but also thinking processes and motivation. Individuals with high self-efficacy are less likely to avoid new and challenging experiences and are more determined to complete their actions successfully. Self-efficacy serves as a catalyst for initiating action; regardless of the potential and advantages one possesses, a lack of self-perceived competence in a specific domain can pose difficulties in initiating and sustaining actions (Erol & Avcı Temizer, 2016). Students with low programming self-efficacy may struggle with programming even if they have the necessary knowledge and skills (Aşkar & Davenport, 2009). If their belief in their programming abilities is low and they perceive programming as inherently difficult (Altun & Mazman, 2012), they may experience difficulty and lower performance in programming. Tsai et al. (2019) conceptualized programming self-efficacy into five dimensions: Logical Thinking, Cooperation, Algorithm, Control, and Debug, representing students' beliefs in their ability to write programs using logical conditions, their perception of cooperation in programming tasks, their ability to develop algorithms independently, their sense of control over program editing, and their ability to debug programs, respectively.

When examining the scales adapted or developed to measure programming self-efficacy in Turkey, it can be observed that they are specific to a particular programming language (Altun & Mazman, 2012; Aşkar & Davenport, 2009; Govender & Basak, 2015; Korkmaz & Altun, 2014) or platform-specific (Altun & Kasalak, 2018). In addition to these, there is a programming self-efficacy scale for middle school students consisting of a single dimension (Kukul et al., 2017) and a programming self-efficacy scale for high school students consisting of three sub-dimensions (Cesur Özkara & Yanpar Yelken, 2020). In Turkey, middle school ICT teachers use different programming environments such as Scratch, Kodu Game, Small Basic, Lego Mindstorms, Java, Code.org, Lightbot, Code Monkey, Appinventor, Ardunio for programming education (Yecan et al., 2017). Therefore, there

is a need for scales that are not specific to a particular programming language or platform, but encompass the sub-dimensions of the programming process, in order to measure the programming self-efficacy of middle school students. When examining the relevant literature, it has been determined that the "Computer Programming Self-Efficacy Scale for Computer Literacy Education (CPSES)" developed by Tsai et al. (2019) has the desired features. The CPSES scale has been adapted to Turkish with the participation of high school and university students (Ekici & Çınar, 2020) and only university students (Gökoğlu, 2022). However, no adaptation study has been conducted for the CPSES scale specifically for middle school students. In this context, the aim of this study is to adapt the CPSES developed by Tsai et al. (2019) to Turkish for middle school students and to obtain a valid and reliable measurement tool. For this purpose, how is the validity and reliability of the Turkish adaptation of the CPSES? The answer to the question has been sought.

METHOD

Participants

The participants of the study consisted of middle school students from the central district of a western province in Turkey. The participants were determined using the non-probability sampling method of convenience sampling, which allows for easy access to participants. This method was preferred due to its time and cost efficiency. A total of 348 eighth-grade students voluntarily participated in the study during the fall semester of the 2022-2023 academic year. Of the students, 186 (53%) were female and 162 (47%) were male, and all of them had programming experience.

The Original Measurement Tool

The CPSES was developed by Tsai et al. (2019). The scale consists of five factors and a total of 16 items (Table 1). These factors are Logical Thinking (4 items), Cooperation (3 items), Algorithm (3 items), Control (3 items), and Debug (3 items). The construct validity of the scale was tested using exploratory factor analysis (EFA). After EFA, it was determined that the scale consists of 16 items grouped under five sub-factors and that the factors account for 83.87% of the total variance. The Cronbach's alpha values for the factors and the overall scale were found to be 0.96, 0.95, 0.92, 0.84, 0.94, and 0.96, respectively (Tsai et al., 2019).

Table 1. Sample Example of Sub-Dimensions in The CPSES and Their Cronbach Values

Sub-dimension	Example item	Item Number	Cronbach Alpha
Logical Thinking	I can predict the final result of a program with logical conditions.	4	0.96
Cooperation	I can work with others while writing a program.	3	0.95
Algorithm	I can make use of programming to solve a problem.	3	0.92
Control	I can run and test a program in a program editor.	3	0.84
Debug	I can fix an error while testing a program.	3	0.94

Adaptation of the Scale to Turkish and the Application

To adapt the scale to Turkish culture, permission was first obtained from the authors who developed the scale. Then, a bilingual academician translated the scale into Turkish. In the translation from English to Turkish, semantic and conceptual deductions were made. The opinions of three researchers from the departments of Turkish Language Teaching, Computer and Instructional Technologies Education, and English Language Teaching were obtained regarding the semantic translation, and corrections were made in three items. Subsequently, the resulting Turkish version was back-translated into English by another bilingual academician who had not seen the original English version. Differences between the versions were compared, and items were revised to ensure clarity. Then, two prospective computer and instructional technologies teachers checked the comprehensibility of the Turkish scale by the target audience. After obtaining the opinions of computer and instructional technologies teachers, the final version of the Turkish scale was prepared. To determine the level of participant agreement with the items, a 5-point Likert scale was used, as in the original scale (1 = strongly disagree, 2 = disagree, 3 = neither agree nor disagree, 4 = agree, 5 = strongly agree).

Data Analysis

Since the factor structures of the scale were already known, first and second-order confirmatory factor analyses were conducted to test the structural validity of the CPSES scale. All statistical analyses were performed using the lavaan package (Rosseel, 2012, 2021) in the R software (R Core Team, 2019). Multivariate normality was examined using the Mardia test to determine which estimation algorithm to use (Gana & Broc, 2019). The results of the Mardia test indicated non-normal distribution of multivariate data ($p < .001$). Therefore, the weighted least square mean and variance adjusted (WLSMV) estimator was used instead of the Maximum Likelihood (ML) estimator. The WLSMV estimator is recommended for analyzing categorical data obtained from Likert-type ordinal scales (Brown, 2006; DiStefano & Morgan, 2014).

To assess the goodness of fit for both the measurement model and the structural model, the following fit indices and threshold values were considered: chi-square to degrees of freedom ratio ($\chi 2/df$) < 3.00; comparative fit index (CFI) > .90 (Kline, 2016); Tucker–Lewis's index (TLI) > .90 (Hair et al., 2019); root mean square error of approximation (RMSEA) < .08; and standardized root mean residual (SRMR) < .06 (Hu & Bentler, 1999).

FINDINGS

Descriptive Statistics

Descriptive statistics, reliability values for the subscales of the adapted scale, and the overall scale are presented in Table 2. All mean scores were above the mid-point of 3.000, indicating an overall positive response to the variables in the model. The standard deviations ranged from 0.794 to 1.109, reflecting a moderate spread of students' responses. The Cronbach's alpha values obtained from the subscales of the scale ranged from .797 to .854, while the overall scale reliability value was .909. These values are above the threshold value of .70, indicating that the adapted scale is reliable.

Table 2. Descriptive Statistics and Cronbach Alpha Reliability Values for the Adapted Scale

Sub-dimension	Item Number	M	SD	Cronbach Alpha
Logical Thinking	4	3.122	0.980	.809
Cooperation	3	3.862	1.003	.854
Algorithm	3	3.193	0.986	.782
Control	3	3.494	1.109	.849
Debug	3	3.521	1.015	.797
Total	16	3.419	0.784	.909

Findings Regarding the Construct Validity of the Adapted Scale

Confirmatory factor analysis (CFA) was used to test the factorial validity of the adapted scale, which consists of 5 factors and 16 items, as the factor structures were previously known. The reliability and convergent validity of the adapted scale were analyzed by examining the average variance extracted (AVE), composite reliability (CR), and Cronbach's alpha values (Table 3). For each factor, an AVE value greater than .50, CR values greater than .70, and Cronbach's alpha value greater than .70 indicate high reliability and achieved convergent validity

Table 3. Result of Reliability and Convergent Validity of the Adapted Scale

Construct	Item	Standardized factor loading	Cronbach Alpha	Composite reliability	Average variance extracted
Logical Thinking	LC1	0.72	0.809	0.807	0.513
	LC2	0.62			
	LC3	0.74			
	LC4	0.79			
Cooperation	CO1	0.94	0.854	0.843	0.643
	CO2	0.71			
	CO3	0.76			
Algorithm	ALG1	0.76	0.782	0.783	0.546
	ALG2	0.68			
	ALG3	0.78			
Control	CON1	0.76	0.849	0.849	0.652
	CON2	0.83			
	CON3	0.83			
Debug	DEB1	0.79	0.797	0.802	0.577
	DEB2	0.79			
	DEB3	0.68			

The discriminant validity of the adapted scale was examined by comparing the squared AVE values for each latent construct with the correlation values with other latent constructs (Table 4). The diagonal values shown in parentheses (square roots of AVE) being higher than the corresponding row and column values indicate that discriminant validity is achieved.

Table 4. Convergent Validity Results of the Adapted Scale

	LT	CO	ALG	CON	DEB
LT	(.716)				
CO	.472	(.802)			
ALG	.690	.523	(.739)		
CON	.633	.458	.640	(.807)	
DEB	.633	.598	.740	.678	(.760)

Note. LT = Logical Thinking, CO = Cooperation, ALG = Algorithm, CON = Control, DEB = Debugging

According to the First Order CFA results (Fig.1), the model had a good fit with the data obtained from middle school students ($\chi2$ = 177.344, df = 120, p < .05; $\chi2$/df = 1.887; CFI = .949; TLI = .935; RMSEA = .051 CI [.039, .062]; SRMR = .043). The item factor loadings of the scale ranged from 0.68 to 0.94.

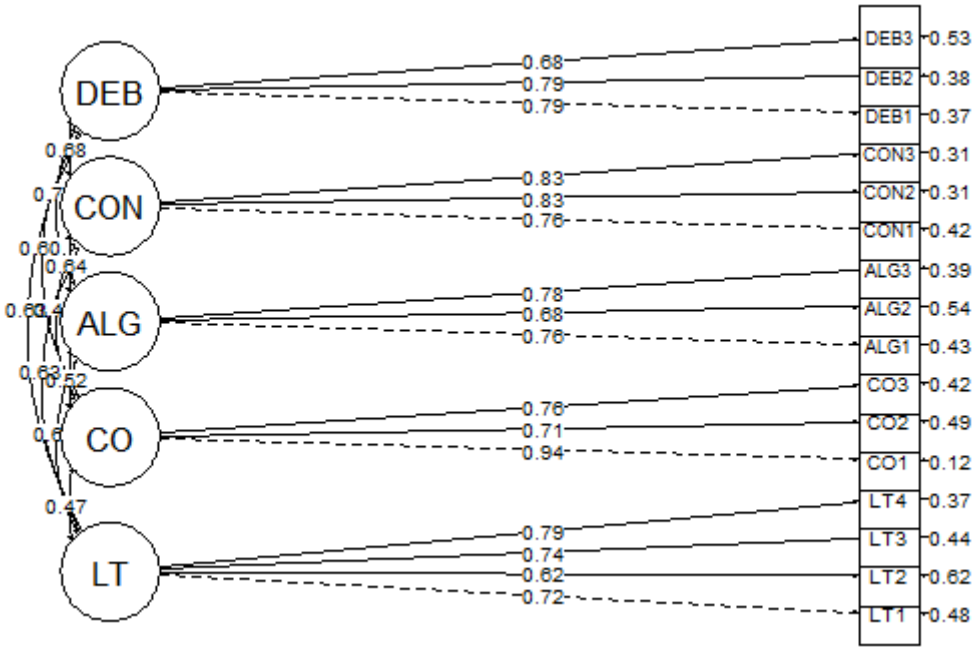

Figure 1. First order CFA results

In addition to the first-order CFA, a second-order CFA was performed to determine the extent to which the latent constructs of debugging, control, algorithm, cooperation, and logical thinking fit the latent construct of "Programming Self-efficacy" defined as a superstructure (Fig. 2). According to the second level CFA results, the model had good fit values ($\chi2$ = 365.555, df = 99, p < .05; $\chi2$/df = 1.672; CFI = .959; TLI = .951; RMSEA = .044 CI [.032, .056]; SRMR = .046). These values showed that the factor structure of the scale was compatible with the data and were accepted as evidence for the construct validity of the scale. This finding means that Programming Self-efficacy can be measured with a five-factor structure named Debugging, Control, Algorithm, Cooperation and Logical Thinking. The fit indices obtained from the first and second level CFA indicate that the construct validity of the Turkish form of the CPSES scale was achieved. All these results show that the adapted CPSES scale consisting of 16 items with 5 factors is reliable and valid and suitable for Turkish culture.

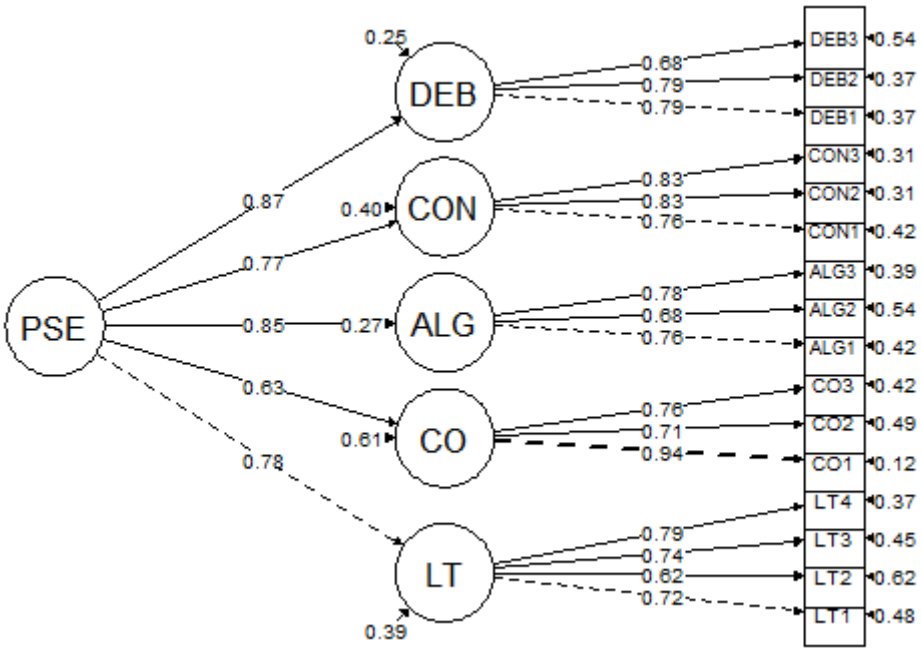

Figure 2. Second order CFA results

CONCLUSION

The aim of this study was to adapt the "Computer Programming Self-Efficacy Scale for Computer Literacy Education (CPSES)" developed by Tsai et al. (2019) into Turkish for middle school students and to establish a valid and reliable measurement tool. The study included 348 eighth-grade students. Since the factor structure of the adapted scale and the association of each item with its respective factor were already known, the factor structures of the scale were tested using confirmatory factor analysis (CFA). The results of both the first and second-order CFAs showed good fit values with the data.

The results of the analysis showed that the reliability of the adapted scale was high and that convergent validity and discriminant validity were achieved. The results of the first-order and second-order confirmatory factor analyses showed that both models had good fit values with the data. All 16 items and 5-factor structure (Logical Thinking, Cooperation, Algorithm, Control, Debug) in the original scale were retained in the Turkish scale. As a result, it was concluded that the Computer Programming Self-Efficacy Scale for Computer Literacy Education (CPSES) is a valid and reliable measurement tool in Turkish culture.

The CPSES scale has four unique features (Tsai et al., 2019). First, the scale is not intended for computer engineering students, but is suitable for anyone at the middle school level and above who has programming experience. Second, the scale does not contain statements specific to a particular programming language or block-based programming environment. Therefore, it can be used to measure programming self-efficacy in general. The third is that the scale has a social dimension, i.e., a collaboration subscale. Lastly, the Control and Debug are the two subscales relating to the self-efficacies of self-regulation in programming. Because of these features, the scale is recommended to be used to measure the programming self-efficacy of middle school students with programming experience and higher-level students and adults regardless of programming language or environment. The CPSES scale can also be used by researchers who want to analyze the sub-dimensions of programming self-efficacy.

The findings of the study provide evidence for the validity and reliability of the CPSES scale developed by Tsai et al. (2019) and show that it is suitable for Turkish culture for middle school students. Therefore, the CPSES scale can be used in future studies to measure programming self-efficacy of middle school students. For future studies, it is recommended that the CPSES scale, which has sufficient validity and reliability evidence, should be applied in different samples with measurement invariance.

REFERENCES

Aktunc, O. (2013). A teaching methodology for introductory programming courses using Alice. *International Journal of Modern Engineering Research*, *3*(1), 350–353.

Altun, A., & Kasalak, İ. (2018). Blok temelli programlamaya ilişkin öz-yeterlik algısı ölçeği geliştirilmesi: Scratch örneği [Perceived self-efficacy scale development study related to block based programming: Scratch case]. *Eğitim Teknolojisi Kuram ve Uygulama*, *8*(1), 209–225.

Altun, A., & Mazman, S. G. (2012). Programlamaya ilişkin öz yeterlilik algısı ölçeğinin Türkçe formumun geçerlilik ve güvenirlik çalışması [Validity and reliability study of the Turkish form of the self-efficacy perception scale for programming]. *Eğitimde ve Psikolojide Ölçme ve Değerlendirme Dergisi*, *3*(2), 297–308. https://dergipark.org.tr/tr/download/article-file/65965

Aşkar, P., & Davenport, D. (2009). An investigation of factors related to self-efficacy for JAVA programming among engineering students. *The Turkish Online Journal of Educational Technology*, *8*(1), 26–32. http://www.tojet.net/volumes/v8i1.pdf#page=27

Bandura, A. (1997). *Self-efficacy: The exercise ofcontrol*. W. H. Freeman and Company.

Başer, M. (2013). Developing attitude scale toward computer programming. *The Journal of Academic Social Science Studies*, *6*(6), 199–215. https://doi.org/10.9761/JASSS1702

Brennan, K., & Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. *Annual American Educational Research Association Meeting, Vancouver, BC, Canada*, 1–25. https://doi.org/10.1.1.296.6602

Brown, T. A. (2006). *Confirmatory factor analysis for applied research*. Guilford Press.

Cesur Özkara, E., & Yanpar Yelken, T. (2020). Ortaöğretim öğrencilerine yönelik programlama öz yeterlik ölçeğinin geliştirilmesi: Geçerlik ve güvenirlik çalışması [Programming self-efficacy scale for high school students: Development, validition and reliability]. *Eğitim Teknolojisi Kuram ve Uygulama*, *10*(2), 345–365.

Çetin, İ., & Özden, M. Y. (2015). Development of computer programming attitude scale for university students. *Computer Applications in Engineering Education*, *23*(5), 667–672. https://doi.org/10.1002/cae.21639

Chen, G., Shen, J., Barth-Cohen, L., Jiang, S., Huang, X., & Eltoukhy, M. (2017). Assessing elementary students' computational thinking in everyday reasoning and robotics programming. *Computers & Education*, *109*, 162–175. https://doi.org/10.1016/j.compedu.2017.03.001

Clements, D. H. (1999). The future of educational computing research: The case of computer programming. *Information Technology in Childhood Education Annual*, 147–179.

Clements, D. H., & Gullo, D. F. (1984). Effects of computer programming on young children's cognition. *Journal of Educational Psychology*, *76*(6), 1051–1058. https://doi.org/10.1037/0022-0663.76.6.1051

Cooper, S., Dann, W., & Pausch, R. (2000). Developing algorithmic thinking with Alice. *Isecon*, *17*, 506–539. http://www.stanford.edu/~coopers/alice/isecon00.PDF

Courte, J., Howard, E. V, & Bishop-Clark, C. (2006). Using Alice in a computer science survey course. *Information Systems Education Journal*, *4*(87), 3–7.

Dann, W., Cooper, S., & Pausch, R. (2000). *Making the connection: Programming with animated small world*. 41–44. https://doi.org/10.1145/343048.343070

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

DiStefano, C., & Morgan, G. B. (2014). A comparison of diagonal weighted least squares robust estimation techniques for ordinal data. *Structural Equation Modeling*, *21*(3), 425–438. https://doi.org/10.1080/10705511.2014.915373

Ekici, M., & Çınar, M. (2020). Bilgisayar programlama öz-yeterlik ölçeğinin Türkçe formunun geçerlik ve güvenirlik çalışması [The validity and reliability study of the Turkish version of computer programming self-efficacy scale]. *Anadolu Journal Of Educational Sciences International*, *10*(2), 1017–1040. https://doi.org/10.18039/ajesi.725161

Erol, M., & Avcı Temizer, D. (2016). Eyleme geçiren bir katalizör "Öz yeterlik algisi": Üniversite öğrencileri üzerine bir inceleme [A catalyst that put into action "perception of self-efficacy": A study on university students]. *Hacettepe Egitim Dergisi*, *31*(4), 711–723. https://doi.org/10.16986/HUJE.2015014223

Ersoy, H., Madran, R. O., & Gülbahar, Y. (2011). Programlama dilleri öğretimine bir model önerisi: Robot programlama [A model proposal for teaching programming languages: Robot programming]. *Akademik Bilişim'11 - XIII. Akademik Bilişim Konferansı Bildirileri 2 - 4 Şubat 2011 İnönü Üniversitesi, Malatya*, 731–736.

Falloon, G. (2016). An analysis of young students' thinking when completing basic coding tasks using Scratch Jnr. On the iPad. *Journal of Computer Assisted Learning*, *32*(6), 576–593. https://doi.org/10.1111/jcal.12155

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

Gana, K., & Broc, G. (2019). *Structural equation modeling with lavaan*. ISTE & Wiley.

Gökoğlu, S. (2022). Bilgisayar okuryazarlığı eğitimi için bilgisayar programlama öz yeterlik ölçeği: Türkçe geçerlik ve güvenirlik çalışması [Computer programming self-efficacy scale for computer literacy education: Turkish validity and reliability study]. *Bolu Abant İzzet Baysal Üniversitesi Eğitim Fakültesi Dergisi*, *22*(2), 529–551. https://doi.org/10.17240/aibuefd.2022..-654547

Govender, D. W., & Basak, S. K. (2015). An investigation of factors related to self-efficacy for Java programming among computer science education students. *Journal of Governance and Regulation*, *4*(4), 612–619. https://doi.org/10.22495/jgr_v4_i4_c5_p6

Grover, S., & Pea, R. (2013a). Computational thinking in K-12: A review of the state of the field. *Educational Researcher*, *42*(1), 38–43. https://doi.org/10.3102/0013189X12463051

Grover, S., Pea, R., & Cooper, S. (2016). Factors influencing computer science learning in middle school. *Proceedings of the 47th ACM Technical Symposium on Computing Science Education - SIGCSE '16*, *March*, 552–557. https://doi.org/10.1145/2839509.2844564

Guzdial, M., & Soloway, E. (2002). Teaching the Nintendo generation to program. *Communications of the ACM*, *45*(4), 17. https://doi.org/10.1145/505248.505261

Gunbatar, M. S., & Karalar, H. (2018). Gender differences in middle school students' attitudes and self-efficacy perceptions towards mBlock programming. *European Journal of Educational Research*, *7*(4), 925-933. https://doi.org/10.12973/eu-jer.7.4.923

Hagge, J. (2017). Scratching beyond the surface of literacy. *Gifted Child Today*, *40*(3), 154–162. https://doi.org/10.1177/1076217517707233

Hair, J. F., Blacks, W. C., Babin, B. J., & Anderson, R. E. (2019). *Multivariate data analysis* (8th ed.). Cengage Learning.

Hu, L. T., & Bentler, P. M. (1999). Cutoff criteria for fit indexes in covariance structure analysis: Conventional criteria versus new alternatives. *Structural Equation Modeling*, *6*(1), 1–55. https://doi.org/10.1080/10705519909540118

Jenkins, T. (2002). On the difficulty of learning to program. *Third Annual Conference of the LTSN-ICS*, 53–58.

Kafai, Y. B., & Burke, Q. (2013). Computer programming goes back to school. *Phi Delta Kappan*, *95*(1), 61–65. https://doi.org/10.1177/003172171309500111

Karalar, H. (2019). Ortaokul öğretmenlerinin fiziksel programlamaya yönelik algıları ve deneyimleri [Middle school teachers' perceptions and experiences towards physical computing]. *Gazi Eğitim Bilimleri Dergisi, 5*(Özel Sayı), 140-156. https://dx.doi.org/10.30855/gjes.2019.os.01.008

Kazakoff, E. R., & Bers, M. U. (2014). Put your robot in, put your robot out: Sequencing through programming robots in early childhood. *Journal of Educational Computing Research*, *50*(4), 553–573. https://doi.org/10.2190/EC.50.4.f

Kelleher, C., & Pausch, R. (2005). Lowering the barriers to programming: A survey of programming environments and languages for novice programmers. *Science*, *37*(2), 83–137. https://doi.org/10.1145/1089733.1089734

Kline, R. B. (2016). *Principles and practice of structural equation modeling* (4th ed.). The Guilford Press.

Korkmaz, Ö., & Altun, H. (2013). Engineering and CEIT student's attitude towards learning computer programming. *International Journal OfSocial Science*, *6*(2), 1169–1185.

Korkmaz, Ö., & Altun, H. (2014). Adapting computer programming self-efficacy scale and engineering students' self-efficacy perceptions. *Participatory Educational Research*, *1*(1), 20–31. https://doi.org/10.17275/per.14.02.1.1

Kukul, V., Gökçearslan, Ş., & Günbatar, M. S. (2017). Computer programming self-efficacy scale (CPSES) for secondary school students: Development, validation and reliability. *Eğitim Teknolojisi Kuram ve Uygulama*, *7*(1), 158–179. https://dergipark.org.tr/tr/download/article-file/272743

Lee, A. (2015). Determining the effects of computer science education at the secondary level on STEM major choices in postsecondary institutions in the United States. *Computers and Education*, *88*, 241–255. https://doi.org/10.1016/j.compedu.2015.04.019

Lee, Y. (2011). Empowering teachers to create educational software : A constructivist approach utilizing Etoys , pair programming and cognitive apprenticeship. *Computers & Education*, *56*(2), 527–538. https://doi.org/10.1016/j.compedu.2010.09.018

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. https://doi.org/10.1016/j.chb.2014.09.012

Manches, A., & Plowman, L. (2015). Computing education in children's early years: A call for debate. *British Journal of Educational Technology*, n/a-n/a. https://doi.org/10.1111/bjet.12355

Moreno-León, J., & Robles, G. (2016). Code to learn with Scratch? A systematic literature review. *2016 IEEE Global Engineering Education Conference (EDUCON)*, *10-13-Apri*(April), 150–156. https://doi.org/10.1109/EDUCON.2016.7474546

Palumbo, D. B. (1990). Programming language/problem-solving research: A review of relevant issues. *Review of Educational Research*, *60*(1), 65–89. https://doi.org/10.3102/00346543060001065

Papert, S. (1980). *Mindstorms children, computers, and powerful ideas*. Basic Books, Inc.

Pausch, R., Dann, W., & Cooper, S. (2000). Alice: A 3-D tool for introductory programming concepts. *Journal of Computing Sciences in Colleges*, *15*(5), 107–116.

R Core Team. (2019). *R: A Language and environment for statistical computing* (Version 3.6) [Computer software]. https://cran.r-project.org/

Resnick, M. (2002). Rethinking learning in the digital age. In G. Kirkman (Ed.), *The global information technology report: Readiness for the networked world* (pp. 32–37). Oxford University Press. https://doi.org/10.1353/cj.2007.0001

Resnick, M. (2012). Point of View: Reviving Papert's Dream. *Educational Technology*, *52*(4), 42–46.

Resnick, M., Silverman, B., Kafai, Y., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., & Silver, J. (2009). Scratch: Programming for all. *Communications of the ACM*, *52*(11), 60. https://doi.org/10.1145/1592761.1592779

Rosseel, Y. (2012). Lavaan: An R package for structural equation modeling. *Journal of Statistical Software*, *48*(2). https://doi.org/10.18637/jss.v048.i02

Rosseel, Y. (2021). *The lavaan tutorial*. https://lavaan.ugent.be/tutorial/tutorial.pdf

Saeli, M., Perrenet, J., Jochems, W. M. G., & Zwaneveld, B. (2011). Teaching programming in secondary school : A pedagogical content knowledge perspective. *Informatics in Education*, *10*(1), 73–88. http://www.mii.lt/informatics_in_education/pdf/infe177.pdf

Tsai, M. J., Wang, C. Y., & Hsu, P. F. (2019). Developing the computer programming self-efficacy scale for computer literacy education. *Journal of Educational Computing Research*, *56*(8), 1345–1360. https://doi.org/10.1177/0735633117746747

Utting, I., Cooper, S., Kölling, M., Maloney, J., & Resnick, M. (2010). Alice, greenfoot, and scratch -- a discussion. *ACM Transactions on Computing Education*, *10*(4), 1–11. https://doi.org/10.1145/1868358.1868364.http

Wilson, C., Sudal, L. A., Stephenson, C., & Stehlik, M. (2010). *Running the empty: Failure to teach K-12 computer science in the digital age*. Association for Computing Machinery.

Yecan, E., Özçınar, H., & Tayfun, T. (2017). Bilişim teknolojileri öğretmenlerinin görsel programlama öğretimi deneyimleri [ICT teachers' visual programming teaching experiences]. *İlköğretim Online*, *16*(1), 377–393.