Pre-Service Information Technologies Teachers' Views on Computer Programming Tools for K-12 Level

Serhat Altıok[1]
Erman Yükseltürk[1]

[1]Kırıkkale University

DOI: **10.21585/ijcses.v2i3.28**

Abstract

The purpose of the study is to analyze pre-service IT teachers' views on a five day seminar which is related to current methodologies and tools in K-12 computer programming education. The study sample consisted of 44 pre-service IT teachers who study as 3rd or 4th undergraduate program at Department of Computer Education and Instructional Technology in 21 different universities. The data is collected through a Students' Perceptions about Kid's Programming Language Questionnaire consisting of 27 five-point Likert-type items, grouped under three factors. The collected quantitative data were analyzed using descriptive statistics such as means, standard deviations. The results of the study indicated that almost all visual programming tools have positive effects on students' views, Small Basic is not as effective as other tools. It could be concluded that Small Basic tool is text-based in contrast to the other block-based features.

Keywords: Programming Education, Visual Programming Tools, Pre-Service IT Teachers, Scratch, Small Basic, Alice, App Inventor

1. Introduction

In recent years, best practices of technologies have found new audiences with increasingly children from smartphones and tablet computers to electronic learning toys (Bers, Flannery, Kazakoff, & Sullivan, 2014). Therefore, the children in the 21st century children need to be versatile and adaptable not only modern and future technologies but also need to improve the ability of understand and work with these technologies (Saeli, Perrenet, Jochems, & Zwaneveld, 2011). In other words, it is expected that today's children have the knowledge and skills about these technologies and also use them effectively in their life (Günüç, Odabaşı & Kuzu 2013). In addition to these knowledge and abilities, today's children are expected to have basic skills such as critical thinking, analyzing and synthesizing ability, some requirements related to the rapid development of technology such as information literacy, media literacy, technology literacy and code literacy etc. and personal qualities work collaboratively, being innovative and being productive (Kay & Greenhill, 2011). Programming is one of the common techniques that provides students to developing these knowledge, skills, requirements and competencies for solving real-world problems of the 21st century (Grover & Pea, 2013). There are several research studies in the literature which demonstrate the importance of programming since it enables children to become active producers of interactive digital environments and its positive effects (Fesakis & Serafeim, 2009; Kalelioğlu & Gülbahar, 2014) on academic success (Fessakis, Gouli, & Mavroudi, 2013), problem solving performance (Fesakis & Serafeim, 2009) and building children's computational thinking skills etc. (Bers et al., 2014; Brennan & Resnick, 2012).

Despite the benefits and importance of computer programming, it is considered to be difficult to master and understand the core concepts. The reason of this consideration is programming performed in several steps such as generate a solution to a problem, reflect on how to communicate the solution to the machine and using syntax and grammar through an exact way of thinking (Szlávi & Zsakó, 2006). In other words, programming performed in three steps (Pears et al., 2007): problem solving, learning a particular programming language and code/system production. Most students have difficulties in these steps of programming (Lister et al., 2004; Robins, Rountree, & Rountree, 2003). There are also numerous difficulties for students in programming learning and teaching in the literature (Du Boulay, 1986):

- Orientation (What programming is useful for and what the benefits to learn to program are)
- The notional machine (Understanding the general properties of the machine and how the behavior of the physical machine relates to the notional machine)
- Notation (Includes the problems of aspects of the various formal languages such as syntax and semantics)
- Structures (The schemas or plans that can be used to reach small-scale goals (e.g., using a loop))

Moreover, studies found in literature show that many problems in learning programming originate from abstract and complexity of the concepts such as variables, loops, arrays, functions, and syntax of programming languages. These difficulties may become barriers for learning programming skills (Ozoran, Cagiltay, & Topalli, 2012), especially for novices.

Actually, all programmers are novice at the beginning, and it has to know that learning to programming is hard and has to overcome from a wide range of difficulties and deficits (Winslow, 1996). However, turn a novice into an expert programmer takes roughly 10 years (Winslow, 1996) and this continuum has five breakdowns into stages (Dreyfus, 1986): novice, advanced beginner, competence, proficiency, and expert. Just because an expert must be mastered on five specialties (Du Boulay, 1986): general program knowledge, hardware components and their relationship with programs, syntax and semantics of a particular programming language, structures (e.g., schemas, plans), and approaches or theories (that includes continuums such as planning, developing, testing and debugging etc.)

Unlike the experts, novices limited superficially knowledge, lack detailed mental models, fail to apply relevant knowledge, and approach programming ‘‘line by line’’ rather than using meaningful program ‘‘chunks’’ or structures (Winslow, 1996). Novice programmers hold misunderstanding and misconceptions (Saeli et al., 2011). Thus, novices’ incomplete prior knowledge such as misconceptions, deficits in planning, developing and testing of code can be a source of errors, and more (Spohrer & Soloway, 1989). This information shows that students should be coached in the process of programming and teaching programming in early steps in personal training should be provided with facilitating methods and tools.

1.1. Programming Education

A simple definition of programming is the process of writing, testing, debugging/troubleshooting, and maintaining the source of code of computer programs (Wikipedia, 2017). Another definition of programming is the process of developing and implementing various sets of instructions to enable a computer to perform a certain task, solve problems, and provide human interactivity (ECDL Foundation, 2015). Similarly, Moström (2011, p9) defined that “programming is the act of understanding a problem, formulating a solution, and writing down the solution in such a way that a computer can use the solution to solve the problem”. Saeli et al. (2011) analyzed what the reasons to teach programming and their results are enhancing students’ problem solving skills and offering the students a subject, which includes aspects of different disciplines; use of modularity and transferability of the knowledge/skills; and the opportunity to work with a multi-disciplinary subject.

Programming education that has been realized and researched extensively through different methods, languages, tools and methodologies at different levels from primary education to university level have been increased its importance more and more every day. In recent years, there have been a growing number of countries which are focusing their Information and Communications Technology (ICT) curricula on developing students’ computer programming and coding skills that facilitates building higher-order thinking skills. For example, computer programming and coding became an important part of the curriculum in 12 Europe countries: Bulgaria, Cyprus, Czech Republic, Denmark, Estonia, Greece, Ireland, Italy, Lithuania, Poland, Portugal and the UK (Johnson et al., 2014). Similarly, eight countries integrate coding in the ICT curriculum, some countries such as United Kingdom, Estonia, Greece, and Lithuania integrate programming not only in the general ICT course, but also as a specific standalone course in primary curricula (Johnson et al., 2014). Despite most of the countries introducing computing in K-12 curriculum as a whole, some countries in Europe have been applied introducing computing in either K–9 or grades 10–12 (Heintz, Mannila, & Färnqvist, 2016).

Moreover, well-known efforts in the US are the “Computer Science Principles” that aims to develop effective high school computing curricula enacted in 10,000 high schools taught by 10,000 well-prepared teachers by 2016, “The Beauty and Joy of Computing” that exposes students to the beauty and joy of programming by engaging them in meaningful projects using the Snap! Programming language, and “Code.org” that is a high

school course with lessons and programming projects (Angeli et al., 2016; Astrachan, Briggs, Diaz, & Osborne, 2013; desJardins, 2015). The main purpose of introducing computing in primary education is to produce thinkers, as opposed to coders (Repenning, Basawapatna, & Escherle, 2016). In spite of the fact that the coders write codes in any programming languages for solving any problems through produce software that can be divide into application and system, thinkers develop patterns to solve all problems of that type instead of solving any problem by produce generalizable solutions (Selby & Woollard, 2013).

In Turkey, primary education involves core and track subjects at first and second level in primary education such as mathematics, science, social science, language and communication (Ministry of Education, 2012; Sağlam, 2014). Although these lecture-based subjects provide to students knowledge and skills (e.g. thinking, understanding and reasoning), they need to developed higher-order thinking skills such as critical, logical, reflective, metacognitive, and creative thinking (King, Goodson, & Rohani, 2010). For improve these skills, not only should core subjects should be integrated with each other but also Information Technologies (IT) lesson with properties such as student-centered, problem-based, or project-based must be utilized more effectively. Because of this requirement, Ministry of National Education of Turkey initially updated to Information Technologies course as Information Technologies and Software course in 2012, after defined IT lesson that was an elective subject as a compulsory subject at 5th and 6th grade levels of primary education in 2013. Moreover, a regulatory commission was formed by the ministry for include more coding training into the IT lesson curriculum that is compulsory at 5th and 6th grades and elective at 7th and 8th grade for developing algorithmic and computational thinking skills in students. In addition to public institutions, non-governmental organizations have also implemented various projects and activities in order to provide programming and software skills to the students (e.g. Informatics Association of Turkey (IAT) organized an event entitled "Computer Programming is as Easy as Pie " in May, 2014). Association of Information Technology Educators have been realized numerous organizations that including coding, robotics and physical programming trainings to teachers of Information Technologies for bringing to students more programming skills (e.g. Manisa City is Coding, Antalya City is Coding etc.) in recent years. These organizations have been carried through the contributions of the academicians in the departments of Computer Education and Instructional Technology of the universities in these cities. Additionally, academicians in this department realized numerous seminars and workshops through the agency of support from public institutions to introduce alternative pedagogic approaches and visual programming tools that are used for teaching programming to students by teachers or pre-service teachers in recent years (e.g. "How to Teach Programming to Children" seminar and “Programming my Own Game” seminar, CEIT, Kırıkkale University). Besides all these institutional efforts, thousands of students from all grades of education from primary school to university participate individually web-based organizations such as “All Can Code”, “Hour of Code”, “Code Monkey” etc.

1.2. New Applications for Programming Education

Visual programming affect novice programmers' performance and require them to manipulate visual elements to formulate and test of problem (Gouws, Bradshaw, & Wentworth, 2013; Maloney, Resnick, Rusk, Silverman, & Eastmond, 2010). In recent years, there are several visual programming tools allows users who has limited or no programming background to create interactive media-rich projects such as games, simulations, and animations. These tools have demonstrated their particular benefits to assist learning programming and problem solving (Lye & Koh, 2014), and help novice programmers to construct their programs and understand the process of program execution (Kelleher & Pausch, 2005). In brief, visual programming tools help novice programmers to improve programming skills and allows creating and demonstrating digital artifacts through problem solving strategies (Lye & Koh, 2014). Therefore, many visual programming tools available provide children to develop different types programs such as games, animations, interactive stories, mobile applications, or robotic applications. Researchers identified 113 different visual programming tools in many different types in the literature such as block-based, text-based or tile-based.

- Block-based visual programming tools (e.g., Scratch) allow users to construct scripts by dragging-and-dropping code blocks and provide visual feedback to comprehend how code blocks work (Maloney et al., 2010).
- Text-based visual programming tools (e.g., Small Basic) provide a simplified programming environment with syntax highlighting and code completion facility (Microsoft, 2017).
- Tile-based visual programming tools (e.g., Kodu Game Lab) enable users to create and play video games and animated stories through placement of tiles in a meaningful sequence (Fowler, Fristoe, &

MacLaurin, 2012).

Furthermore, these tools are very different from the various features (e.g. purpose of use, age level, level of difficulty, whether or not paid, pedagogical effectiveness and platform type). Therefore, choosing the most effective and purposeful appropriate visual programming tool is directly related to have deep knowledge about features of these tools. In this study, researchers selected Scratch, Small Basic, Alice, and App Inventor because of their different properties such as types of product platform, coding styles, and 2D/3D structures. Another reason for researchers to choose these visual programming tools is age ranges proposed by the person (s), institution (s) or organization (s) that developed these tools are appropriate. Appropriate age ranges of visual programming tools that recommended by the official producer are shown in Figure 1 below (Microsoft, 2017; MIT, 2017b).

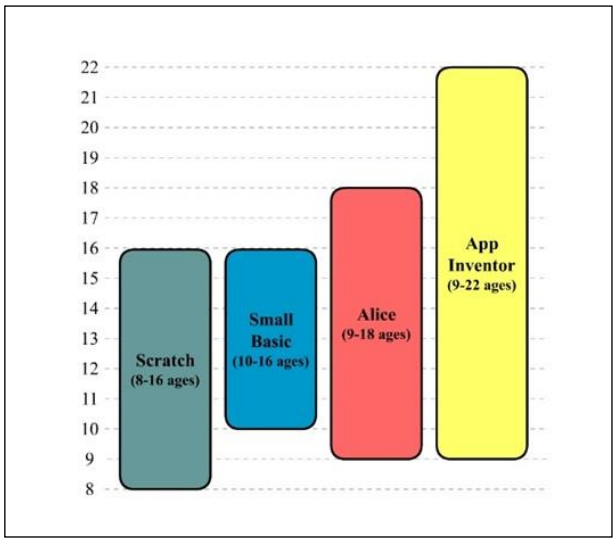

Figure 1. The Appropriate Age Ranges Recommended by the Official Producer of Visual Programming Tools

Although research groups that produce visual programming tools have made suggestions that they are more appropriate at specified age ranges, this age range can be expanded in the direction of need or level of knowledge and use as bidirectional (reducing lower age limit or increasing upper age limit).

1.2.1. Scratch

Scratch (https://scratch.mit.edu/) is a free educational programming tool that was developed by the Lifelong Kindergarten group at the Massachusetts Institute of Technology (MIT) Media Laboratory. The Scratch project began in 2003, and its software and website were publicly launched in 2007. Nowadays, it hosts over 15 million shared projects and almost 80 million comments have been posted about projects by 12 million registered users (MIT, 2017b). The distribution of users' registration age is shown in Figure 2 below.

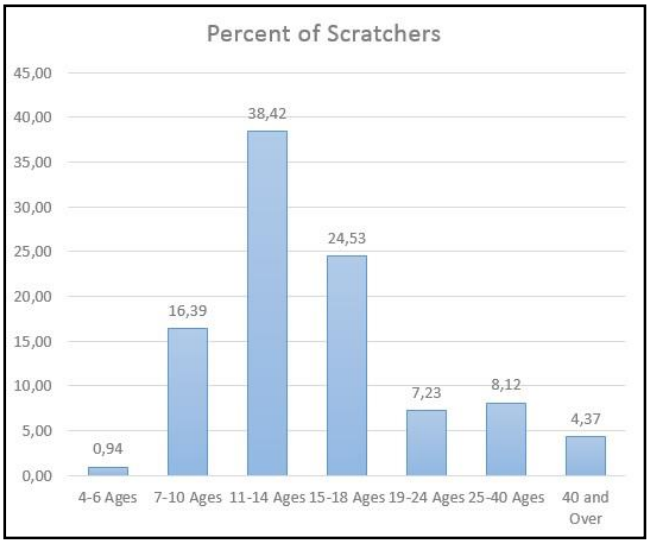

Figure 2. The Distribution of Scratchers' Ages

Scratch provides users create games, animations and simulations by dragging and dropping instead of coding that requires understanding or knowing some concepts such as variables, loops, arrays or functions. Because visual and enjoyable, scratch is preferred by especially young people (MIT, 2017b).

1.2.2. Small Basic

Small Basic (http://smallbasic.com/) is a free educational programming tool that was announced in October 2008, and the first stable version was released in July 2011 by Microsoft. This tool makes programming easy, approachable and funny because it supports conditional branching, loop structures, variables which are weakly typed and dynamic, and subroutines for event handling. Small Basic has two important components: coding screen that allows using codes very simply and a library that provides rich and engaging set of components. This tool also associated with Integrated Development Environment (IDE) which consists of code editor, automation tools and debugger that provides facilitating to software development (Microsoft, 2017).

1.2.3. Alice

Alice (http://www.alice.org) is a free and object-based educational programming tool that was developed on the programming language Python (http://www.python.org) by the Stage 3 Research Group at Carnegie Mellon University led by Randy Pausch in 1997. This tool makes programming easy to learn and create an animation, an interactive game, or a video by drag and drop graphic tiles. By manipulating the objects, Alice also allows to see running of animation, game or a video and facilitates to understand the relationship between programming and objects (Carnegie Mellon University, 1997).

1.2.4. App Inventor

MIT App Inventor (http://appinventor.mit.edu/) is a free and open-source educational programming tool that for designing and building mobile applications that can run on Android devices. App Inventor uses simple graphical interface that very similar to Scratch, which allows drag and drop building to create a basic, fully functional application within an hour or less. This tool was developed by the team was led by Hal Abelson, Mark Friedman, Eric Klopfer and Mitchel Resnick in March 2012. Nowadays, it hosts over 12 million applications have been built by 4 million registered users in 195 countries (MIT, 2017a).

2. Method

This is a descriptive study conducted with a quantitative basis. Descriptive studies, as the name implies, are carried out to describe the characteristics and views of the studied subject (Fraenkel & Wallen, 2009). In this study, the pre-service IT teachers' views were analyzed about computer programming teaching tools especially for K-12 level.

2.1. Participants

The sample of this study was selected from the participants who attended a five day seminar program at a university in Turkey. They were 44 pre-service IT teachers who study as 3rd or 4th undergraduate student at Department of Computer Education and Instructional Technology in 21 different universities.

Table 1. The characteristics of the participants

	N	**%**
Gender		
Male	19	43.18
Female	25	56.82
Weekly Hours of Computer Use		
0-2 hours	9	20.46
3-5 hours	12	27.27

6-8 hours	23	52.27
Computer Programming Knowledge		
Low	21	47.73
Intermediate	17	38.63
High	6	13.64

As it is showed in Table 1, nineteen (43.18%) of them were male and twenty five (56.82%) of them were female. 79.54% of the pre-service IT teachers use computer more than 3 hours in one day. In regard to computer programming knowledge, almost half of the participants (48%) rated their programming knowledge level as low.

2.2. Settings

This is a seminar program for pre-service IT teachers about alternative methods and tools in computer programming for K-12. It was supported by Scientific Meetings Grant Programs. Throughout the program, several seminars were organized to present pre-service IT teachers with pedagogical information specifically for programming instruction for elementary school students and to demonstrate the practical use of current tools used to teach programming to students. The program was started with the introduction of basic pedagogical concepts that have an important place in the education of children and continued with pedagogical examination of how to teach children programming. Later, the tools used for programming teaching were described and the four most common tools used in teaching programming to children in recent years were mentioned. Basic programming principles with Scratch and Small Basic, 3D graphics programming with Alice, Android based mobile programming with App Inventor were discussed with practical examples at computer laboratories. Eight academicians from different universities participated as educators to give seminars for the program. Participants throughout the program were hosted by the University and all costs of the participants (e.g. road, accommodation and meals) were covered.

2.3. Instrumentation

To collect relevant data in this study, researchers used quantitative method. The following instrument helped us to collect quantitative data: Students' Perceptions about Kid's Programming Language Questionnaire (SPKPL-Q). It was developed by Akcay (2009) to obtain the students' perceptions about Small Basic. In this study, it was adapted for Scratch, Alice and App Inventor in addition to Small Basic. It is a 5-point Likert-type scale, consisting of 27 items, grouped under three factors (Perceived Motivation, Perceived Usefulness and Perceived Ease of Use). The Cronbach-Alpha reliability coefficient of the scale was found to be between 0.806 and 0.865.

2.4. Data Collection and Analysis

In the first day of the seminar program, the participants were mentioned about the pedagogy of programming education, programming tools and alternative methodologies. Later, the participants were taught about Scratch, Small Basic, Alice and App Inventor interface, usage and developed applications. At the end of seminars, researchers collected the quantitative data through the questionnaire which included four programming tools (Scratch, Small Basic, Alice and App Inventor). During analyzing of the collected data, the descriptive statistics such as mean and standard deviations of pre-service IT teachers' views about four programming tools were calculated based on the SPKPL-Q scale scores. Also, one way analysis of variance test (ANOVA) was conducted to test the mean differences of pre-service IT teachers' views about four programming tools. ANOVA was considered to be appropriate for the analysis of the data in the study because there is an analysis method which is used to test whether the difference between the averages of two or more unrelated samples is significantly different from zero (Büyüköztürk, 2004). Before the analysis of results, the assumptions of ANOVA have been tested. Each group has a normal distribution and the variances of the groups are homogenized ($p>0.05$).

3. Results

The findings of this study related with pre-service IT teachers' views about four programming tools are given in the Table 2 and Table 3. According to the Table 2, the percentage strongly agree or agree of pre-service IT

teachers' views about Scratch programming in regard to perceived motivation, perceived usefulness and perceived ease of use were 92.2, 94.7 and 92.6 respectively. The percentage strongly agree or agree of pre-service IT teachers' views about Small Basic programming in regard to perceived motivation, perceived usefulness and perceived ease of use were 77.4, 77.6 and 55 respectively. The percentage strongly agree or agree of pre-service IT teachers' views about Alice programming in regard to perceived motivation, perceived usefulness and perceived ease of use were 88.1, 90.5 and 87.1 respectively. The percentage strongly agree or agree of pre-service IT teachers' views about App Inventor programming in regard to perceived motivation, perceived usefulness and perceived ease of use were 92.8, 96.2 and 88.2 respectively. According to the results, the pre-service IT teachers' views about programming tools were generally positive. The lowest ratio related to percentage of pre-service IT teachers' views about programming tools were Small Basic programming.

Table 2. The percentages of pre-service IT teachers' views on programming tools

	Scratch					**Small Basic**					**Alice**					**App Inventor**				
	SD	D	N	**A**	**SA**	SD	D	N	**A**	**SA**	SD	D	N	**A**	**SA**	SD	D	N	**A**	**SA**
	%	%	%	**%**	**%**	%	%	%	**%**	**%**	%	%	%	**%**	**%**	%	%	%	**%**	**%**
Perceived Motivation																				
Interest / Enjoyment	0.76	1.52	3.79	**21.97**	**71.97**	3.79	6.06	21.21	**34.85**	**34.09**	2.27	2.27	2.27	**32.58**	**60.61**	1.52	1.52	5.30	**21.97**	**69.70**
Perceived Competence	2.27	0.00	4.55	**31.82**	**61.36**	0.00	6.82	6.82	**54.54**	**31.82**	4.55	0.00	6.82	**36.36**	**52.27**	0.00	0.00	0.00	**36.36**	**63.64**
Willingness	4.54	3.03	1.51	**29.54**	**61.36**	3.79	10.61	14.39	**33.33**	**37.88**	5.30	3.79	6.82	**31.82**	**52.27**	5.30	3.03	0.76	**28.79**	**62.12**
Participation	0.00	2.27	6.82	**37.50**	**53.41**	3.41	3.41	18.18	**39.77**	**35.23**	1.14	1.14	11.36	**38.64**	**47.73**	1.14	2.27	7.95	**30.68**	**57.95**
Average				**30.21**	**62.03**				**40,62**	**34,76**				**34.85**	**53.22**				**29.45**	**63.35**
Perceived Usefulness																				
Work More Quickly	2.27	2.27	1.14	**26.14**	**68.18**	2.27	3.41	15.91	**32.95**	**45.45**	3.41	1.14	2.27	**29.54**	**63.64**	3.41	0.00	1.14	**30.68**	**64.77**
Job Performance	1.14	0.00	3.41	**31.82**	**63.64**	5.68	4.55	7.95	**39.77**	**42.04**	2.27	2.27	3.41	**35.23**	**56.82**	1.14	1.14	1.14	**32.95**	**63.64**
Increase Productivity	0.00	0.00	2.27	**36.36**	**61.36**	2.27	4.55	13.64	**45.45**	**34.09**	2.27	0.00	6.82	**34.09**	**56.82**	0.00	0.00	2.27	**38.64**	**59.09**
Effectiveness	0.00	0.00	6.82	**29.54**	**63.64**	0.00	9.09	18.18	**36.36**	**36.36**	2.27	0.00	9.09	**31.82**	**56.82**	0.00	0.00	6.82	**31.82**	**61.36**
Makes Job Easier	0.00	2.27	2.27	**38.64**	**56.82**	0.00	11.36	15.91	**47.73**	**25.00**	0.00	2.27	13.64	**34.09**	**50.00**	0.00	2.27	0.00	**40.91**	**56.82**
Useful	0.00	1.14	6.82	**34.09**	**57.95**	1.14	5.68	12.50	**42.04**	**38.64**	0.00	0.00	5.68	**43.18**	**51.14**	0.00	0.00	3.41	**38.64**	**57.95**
Average				**32.77**	**61.93**				**40.71**	**36.93**				**34.66**	**55.87**				**35.61**	**60.61**
Perceived Ease of Use																				
Easy to Learn	1.14	1.14	1.14	**39.77**	**56.82**	11.36	17.04	26.14	**23.86**	**21.59**	1.14	4.55	10.23	**46.59**	**37.50**	0.00	3.41	4.55	**45.45**	**46.59**
Easy to Use	2.27	2.27	4.55	**25.00**	**65.91**	4.55	6.82	27.27	**43.18**	**18.18**	0.00	2.27	0.00	**50.00**	**47.73**	0.00	0.00	9.09	**31.82**	**59.09**
Easy to Become Skillful	0.00	0.00	6.82	**40.91**	**52.27**	6.82	9.09	36.36	**25.00**	**22.73**	0.00	4.55	11.36	**43.18**	**40.91**	0.00	2.27	9.09	**38.64**	**50.00**
Clear and Understandable	4.09	5.45	0.91	**29.54**	**60.00**	6.36	10.45	17.78	**30.91**	**34.54**	1.82	8.64	7.27	**35.45**	**46.82**	3.18	5.91	9.55	**28.18**	**53.18**
Average				**33.81**	**58.75**				**30.74**	**24.26**				**43.81**	**43.24**				**36.02**	**52.22**

Note. SD: Strongly Disagree, D: Disagree, N: Neutral, A: Agree, SA: Strongly Agree.

According to the Table 3, the overall means of pre-service IT teachers' views about Scratch programming in regard to perceived motivation, perceived usefulness and perceived ease of use were 4.49, 4.55 and 4.45 respectively. The overall means of pre-service IT teachers' views about Small Basic programming in regard to perceived motivation, perceived usefulness and perceived ease of use were 3.98, 4.04 and 3.54 respectively. The overall means of pre-service IT teachers' views about Alice programming in regard to perceived motivation, perceived usefulness and perceived ease of use were 4.33, 4.42 and 4.24 respectively. The overall means of pre-service IT teachers' views about App Inventor programming in regard to perceived motivation, perceived usefulness and perceived ease of use were 4.51, 4.55 and 4.36 respectively. The post-hoc test results indicated that there is a significant differences about pre-service IT teachers' views about four programming tools ($p<0.05$). In other words, pre-service IT teachers' views were generally positive, but, the means of their views related to Small Basic programming were lower when comparing with other programming tools.

Table 3. The mean differences of pre-service it teachers' views on programming tools

	Scratch		Small Basic		Alice		App Inventor			
	M	**SD**	**M**	**SD**	**M**	**SD**	**M**	**SD**	***F***	***p***
Perceived Motivation										
Interest / Enjoyment	4.63	0.54	3.89	0.83	4.47	0.60	4.57	0.58	11.99	0.00*
Perceived Competence	4.50	0.79	4.11	0.81	4.32	0.96	4.64	0.49	3.70	0.01*
Willingness	4.40	0.62	3.91	0.91	4.22	0.70	4.39	0.63	4.46	0.00*
Participation	4.42	0.59	4.00	0.85	4.31	0.69	4.42	0.66	3.51	0.02*
Overall Mean	**4.49**	**0.64**	**3.98**	**0.85**	**4.33**	**0.74**	**4.51**	**0.59**		
Perceived Usefulness										
Work More Quickly	4.56	0.69	4.16	0.83	4.49	0.66	4.53	0.67	2.96	0.03*
Job Performance	4.57	0.56	4.08	0.89	4.42	0.69	4.57	0.56	4.96	0.00*
Increase Productivity	4.59	0.54	4.05	0.94	4.43	0.82	4.57	0.55	5.23	0.00*
Effectiveness	4.57	0.62	4.00	0.96	4.41	0.84	4.55	0.63	5.03	0.00*
Makes Job Easier	4.50	0.66	3.86	0.93	4.32	0.80	4.52	0.63	7.02	0.00*
Useful	4.49	0.54	4.11	0.85	4.45	0.55	4.55	0.52	4.23	0.01*
Overall Mean	**4.55**	**0.60**	**4.04**	**0.90**	**4.42**	**0.73**	**4.55**	**0.59**		
Perceived Ease of Use										
Easy to Learn	4.50	0.60	3.27	1.16	4.15	0.72	4.35	0.61	20.54	0.00*
Easy to Use	4.50	0.88	3.64	1.01	4.43	0.62	4.50	0.66	11.91	0.00*
Easy to Become Skillful	4.45	0.63	3.48	1.15	4.20	0.82	4.36	0.75	11.73	0.00*
Clear and Understandable	4.36	0.54	3.77	0.86	4.17	0.59	4.22	0.58	6.57	0.00*
Overall Mean	**4.45**	**0.66**	**3.54**	**1.05**	**4.24**	**0.69**	**4.36**	**0.65**		

Note. M: Mean, SD: Standard Deviation, *: $p<0.05$

4. Discussion

Programming is the most functional way for supporting to developing higher-order thinking skills and algorithmic problem-solving skills (Grover & Pea, 2013; Weintrop et al., 2016). There are numerous programming languages in the literature such as Python, C, C++, C#, Java, JavaScript, PHP, Assembly language, Visual Basic .NET, Perl, Delphi, Ruby, Scala, Haskell, Swift etc. (Pierce, 2002). These structured (conventional) programming languages are widely adopted by educators to teach general purposes of any programming (Pears et al., 2007; Robins et al., 2003; Xinogalos, 2012). The first step in all programming languages is to learn or teach what the core elements such as condition, array, loop, variable, constant, and functions are, what they do,

and how they are used. Studies found in literature showed that many problems in learning programming originate from using of these core elements (Choi, 2012). Since learning these core elements is difficult and tedious process, various projects and activities are organized with the support of non-nonprofit companies, non-governmental organizations and governments in order to make students love programming by making coding easy and fun. One of these widespread efforts is the use of visual programming tools (e.g. Scratch, Alice) in programming education. In this study, a seminar program was organized about teaching alternative methods and tools in computer programming for K-12 level in order to reach similar aims. The pre-service IT teachers' views who attended this seminar program were analyzed based on four visual programming tools and the results showed that they have positive views on the use of these programming tools in programming education in terms of motivation, usefulness and ease of use.

In the last decade, there have been developed several visual programming tools that make students more effective producer through some features such as simplified syntax, drag and drop ability to compose programs, immediate execution of commands. Also, these visual programming tools help novice programmers to improve programming skills and allow creating and demonstrating digital artifacts through problem solving strategies (Lye & Koh, 2014). In other words, visual programming environments have been developed like Scratch, Small Basic, Alice, Lego Mindstorm, in order to make them more compatible to information technology beginners to minimize the learning disabilities and difficulties of the programming. Similarly, almost all visual programming tools have positive effects on pre-service IT teachers' views in this study.

Computer programming skills and learning these skills become more important in the 21st century. However, programming lessons which are given by traditional methods do not attract to students attention and various problems occur. These problems cause to check the teaching method used in the programming education. Alternative visual programming tools have been developed to minimize the problems in programming education. Even though the text-based programming is still common method for teaching programming skill, it has several drawbacks. Similarly, Small basic is more unsuccessful than other visual programming tools such as Scratch, Alice, and App Inventor since it contains tasks likewise in the conventional programming in this study. The results showed that scripting has a negative effect on the pre-service teacher's attitudes towards programming when comparing visual programming tools.

5. Conclusion

Programming is becoming increasingly a key competence which will have to be supported generating computer-based solutions for problems such as program, application, animation, simulation, or game by all students since it improves computational thinking skills as well as high-level thinking skills such as developing creative, critical, strategic, analytical, multidimensional, solution-focused. Developing individuals who have these high-level thinking skills that cannot be developed in a short time and develop as experience grows depends on the provision of programming education as long-term and product-focused to individuals from a young age. However, different requirements of programming languages such as syntax, constructs etc. cause problems in developing individual products. Because of these problems, individuals drop out from their programming education. Instructors prefers visual programming tools in programming education to overcome these problems. Especially at primary education level, it is even more difficult to learn the text codes and use the syntax properties of the programming language without any problems that should be written in languages other than native languages because of the low level of foreign language skills at small age levels. In addition to developing the solutions to problems easier with visual programming, independently testable of code blocks for sub-problems that provided with tools like Scratch provides more effective process on develop algorithms for solving sub-problems for primary students.

In recent years, various summer camps, seminars, educational and social projects have been challenged in order to bring several skills to the students at an early age. One of these has been studied in this study. The aim of this seminar program is to develop the knowledge and skills of IT teacher candidates to update the information about programming education for elementary school students in particular, to demonstrate the practical use of current tools used to teach programming to students, to prepare classroom teaching activities using these applications and to adapt them to laboratory activities. This study analyzed pre-service IT teachers' views on this five day seminar which is related to current methodologies and tools in K-12 computer programming education. The research results showed that pre-service IT teachers have positive views on the use of visual programming tools in programming education. However, it is noteworthy that the views on Small Basic are less positive than the other three tools. According to this information, when pre-service IT teachers begin their professional career, it

may be less likely to prefer Small Basic because of its text-based programming requirements. Nevertheless, it is important that pre-service IT teachers will use even three of the four tools which are trained to use as a trainer. If pre-service IT Teachers are taught a greater variety of visual programming tools then they can specialize in the use of one specific tool that can be more appropriate and effective for them. For this reason, pre-service IT teachers should be provided numerous trainings that includes alternative tools and methodologies.

All of these conditions also drives the development of several educational programming tools especially for novices and young students. On the contrary, we have to think that the availability of software programming environments is not enough for the utilization of the learning potential of programming. Experimentally validated teaching/learning approaches, documented best practices, learning resources, curriculum standards, professional development and support for teachers are also needed (Fessakis et al., 2013). In addition, pre-service teachers want to be aware of advanced technology and current pedagogical information and they need training to improve themselves. Increasing the number of such events will be important in terms of updating teacher candidates' information and informing them of new developments. Some potential limitations of this study also should be taken into consideration while discussing the results since only four visual programming tools were analyzed in a seminar program. The study population consisted of only 44 pre-service IT teachers attending in this seminar, which limits the generalizability of the results. Extending the population to various activities, programs and universities could produce different results.

Acknowledgments

This study was conducted with financial support of The Scientific and Technological Research Council of Turkey (Project No: Scientific Meetings Grant Programmes-2229-2016/1).

References

Akçay, T. (2009). Perceptions of students and teachers about the use of a kid's programming language in computer courses. Unpublished MS Thesis. Middle East Technical University, Ankara, Turkey.

Angeli, C., Voogt, J., Fluck, A., Webb, M., Cox, M., Malyn-Smith, J., & Zagami, J. (2016). A K-6 computational thinking curriculum framework: implications for teacher knowledge. *Educational Technology & Society,* 19(3), 47-58.

Astrachan, O., Briggs, A., Diaz, L., & Osborne, R. B. (2013). CS principles: development and evolution of a course and a community. *Paper presented at the Proceeding of the 44th ACM technical symposium on Computer science education.* Retrieved June 15, 2018, from https://doi.org/10.1145/2445196.2445382

Bers, M. U., Flannery, L., Kazakoff, E. R., & Sullivan, A. (2014). Computational thinking and tinkering: Exploration of an early childhood robotics curriculum. *Computers & Education, 72*, 145-157. Retrieved June 15, 2018, from http://doi.org/110.1016/j.compedu.2013.1010.1020

Brennan, K., & Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. *Paper presented at the Proceedings of the 2012 annual meeting of the American Educational Research Association*, Vancouver, Canada. Retrieved June 15, 2018, from http://scratched.gse.harvard.edu/ct/files/AERA2012.pdf

Büyüköztürk, Ş. (2018). *Sosyal Bilimler için Veri Analizi El Kitabı* (24. Baskı). Pegem Yayıncılık, Ankara.

Carnegie Mellon University. (1997). About Alice. Retrieved June 15, 2018, from http://www.alice.org/index.php?page=what_is_alice/what_is_alice

Choi, H. (2012). Learners' reflections on computer programming using Scratch: Korean primary pre-service teachers' perspective. *Paper presented at the 10th International Conference for Media in Education* 2012 (ICoME).

desJardins, M. (2015). Creating AP® CS principles: let many flowers bloom. *ACM Inroads*, 6(4), 60-66. Retrieved June 15, 2018, from http://doi.org/10.1145/2835852

Dreyfus, S. E. (1986). *Dynamic programming The Bellman Continuum* (pp. 13-70): World Scientific.

Du Boulay, B. (1986). Some difficulties of learning to program. *Journal of Educational Computing Research*, 2(1), 57-73. Retrieved June 15, 2018, from http://journals.sagepub.com/doi/pdf/10.2190/2193LFX-2199RRF-2167T2198-UVK2199

Fesakis, G., & Serafeim, K. (2009). Influence of the familiarization with scratch on future teachers' opinions and attitudes about programming and ICT in education. P*aper presented at the ACM SIGCSE Bulletin*. Retrieved June 15, 2018, from http://dl.acm.org/citation.cfm?id=1562957

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. Retrieved June 15, 2018, from http://www.sciencedirect.com/science/article/pii/S0360131512002813

Fowler, A., Fristoe, T., & MacLaurin, M. (2012). Kodu Game Lab: a programming environment. *The Computer Games Journal*, 1(1), 17-28. Retrieved June 15, 2018, from https://pdfs.semanticscholar.org/d998/d996a997e934bc952f996e279037c263781a279035f279037a275467a.pdf

Fraenkel, J., & Wallen, N. (2009). *The nature of qualitative research. How to design and evaluate research in education, seventh edition*. Boston: McGraw-Hill, 420.

Gouws, L. A., Bradshaw, K., & Wentworth, P. (2013). Computational thinking in educational activities: an evaluation of the educational game light-bot. *Paper presented at the Proceedings of the 18th ACM conference on Innovation and technology in computer science education*. Retrieved June 15, 2018, from http://dl.acm.org/citation.cfm?id=2466518

Grover, S., & Pea, R. (2013). Computational Thinking in K–12 A Review of the State of the Field. *Educational Researcher*, 42(1), 38-43. Retrieved June 15, 2018, from http://journals.sagepub.com/doi/abs/10.3102/0013189X12463051

Günüç, S., Odabaşı, H. F., ve Kuzu, A. (2013). 21. yüzyıl öğrenci özelliklerinin öğretmen adayları tarafından tanımlanması: Bir Twitter uygulaması. *Eğitimde Kuram ve Uygulama,* 9(4), 436-455.

Heintz, F., Mannila, L., & Färnqvist, T. (2016). A review of models for introducing computational thinking, computer science and computing in K-12 education. *Paper presented at the Frontiers in Education Conference (FIE), 2016 IEEE*. Retrieved June 15, 2018, from http://doi.org/10.1109/FIE.2016.7757410

Johnson, L., Adams Becker, S., Estrada, V., Freeman, A., Kampylis, P., Vuorikari, R., & Punie, Y. (2014). *Horizon Report Europe: 2014 Schools Edition.* Luxembourg: Publications Office of the European Union, & Austin, Texas: The New Media Consortium.

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

Kay, K. & Greenhill, V. (2011). Twenty-first century students needs 21st century skills. In G. Wan & D. M. Gut (Eds.), *Bringing Schools into the 21st Century* (pp. 41–66). Dordrecht, Germany: Springer. Retrieved June 15, 2018, from https://doi.org/10.1007/978-94-007-0268-4

Kelleher, C., & Pausch, R. (2005). Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM Computing Surveys (CSUR)*, 37(2), 83-137. Retrieved June 15, 2018, from http://dl.acm.org/citation.cfm?id=1089734

King, F., Goodson, L., & Rohani, F. (2010). *Higher order thinking skills: Definition, teaching strategies, assessment.* Publication of the Educational Services Program, now known as the Center for Advancement of Learning and Assessment. Retrieved June 15, 2018, from www.cala.fsu.edu

Lister, R., Adams, E. S., Fitzgerald, S., Fone, W., Hamer, J., Lindholm, M., . . . Seppälä, O. (2004). A multi-national study of reading and tracing skills in novice programmers. *Paper presented at the ACM SIGCSE Bulletin*. Retrieved June 15, 2018, from https://doi.org/10.1145/1044550.1041673

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. Retrieved June 15, 2018, from http://www.sciencedirect.com/science/article/pii/S0747563214004634

Maloney, J., Resnick, M., Rusk, N., Silverman, B., & Eastmond, E. (2010). The scratch programming language and environment. *ACM Transactions on Computing Education (TOCE)*, 10(4), 16. Retrieved June 15, 2018, from http://dl.acm.org/citation.cfm?id=1868363

Microsoft. (2017). About Small Basic. Retrieved June 15, 2018, from, http://smallbasic.com/about.aspx

Ministry of Education. (2012). 12-Year Compulsory Education Questions - Answers. Retrieved June 15, 2018, from http://www.meb.gov.tr/duyurular/duyurular2012/12Yil_Soru_Cevaplar.pdf

MIT, M. I. o. T. (2017a). About App Inventor. Retrieved June 15, 2018, from http://appinventor.mit.edu/explore/about-us.html

MIT, M. I. o. T. (2017b). About Scratch. Retrieved June 15, 2018, from https://scratch.mit.edu/about

Ozoran, D., Cagiltay, N., & Topalli, D. (2012). Using scratch in introduction to programming course for engineering students. *Paper presented at the 2nd International Engineering Education Conference (IEEC2012)*.

Pears, A., Seidman, S., Malmi, L., Mannila, L., Adams, E., Bennedsen, J., . . . Paterson, J. (2007). A survey of literature on the teaching of introductory programming. *ACM SIGCSE Bulletin*, 39(4), 204-223.

Pierce, B. C. (2002). *Types and programming languages*. Retrieved June 15, 2018, from http://robotics.upenn.edu/~bcpierce/tapl/contents.pdf

Repenning, A., Basawapatna, A., & Escherle, N. (2016). Computational thinking tools. *Paper presented at the Visual Languages and Human-Centric Computing (VL/HCC), 2016 IEEE Symposium*. Retrieved June 15, 2018, from http://ieeexplore.ieee.org/abstract/document/7739688/

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

Saeli, M., Perrenet, J., Jochems, W. M., & Zwaneveld, B. (2011). Teaching programming in secondary school: a pedagogical content knowledge perspective. *Informatics in Education-An International Journal*, 10(1), 73-88.

Sağlam, M. (2014). The 4+ 4+ 4 in the Educational Experiences of the the Teachers Teaching the First Grade Students in Turkey: Yozgat City as an Example. *Journal of History School*, 7(18), 377-396.

Selby, C., & Woollard, J. (2013). Computational thinking: the developing definition. Retrieved June 15, 2018, from https://eprints.soton.ac.uk/356481/1/Selby_Woollard_bg_soton_eprints.pdf

Spohrer, J. C., & Soloway, E. (1989). Simulating student programmers. Ann Arbor, 1001, 48-109. Received from http://ijcai.org/Proceedings/189-101/Papers/087.pdf

Szlávi, P., & Zsakó, L. (2006). Programming versus application. *Paper presented at the International Conference on Informatics in Secondary Schools-Evolution and Perspectives*. Retrieved June 15, 2018, from http://doi.org/10.1007/11915355_5

Weintrop, D., Beheshti, E., Horn, M., Orton, K., Jona, K., Trouille, L., & Wilensky, U. (2016). Defining computational thinking for mathematics and science classrooms. *Journal of Science Education and Technology*, 25(1), 127-147.

Wikipedia. (2017). Programming. Retrieved June 15, 2018, from https://en.wikipedia.org/wiki/Computer_programming

Winslow, L. E. (1996). Programming pedagogy—a psychological overview. *ACM SIGCSE Bulletin*, 28(3), 17-22.

Xinogalos, S. (2012). An evaluation of knowledge transfer from microworld programming to conventional programming. *Journal of Educational Computing Research, 47*(3), 251-277.