



ISSN: 2148-9955

International Journal of Research in Education and Science (IJRES)

www.ijres.net

Teaching GUI-Programming Concepts to Prospective K12 ICT Teachers: MIT App Inventor as an Alternative to Text-Based Languages

Can Mihci, Nesrin Ozdener Donmez
Marmara University

To cite this article:

Mihci, C. & Ozdener Donmez, N. (2017). Teaching GUI-programming concepts to prospective K12 teachers: MIT app inventor as an alternative to text based languages. *International Journal of Research in Education and Science (IJRES)*, 3(2), 543-559. DOI:10.21890/ijres.327912

This article may be used for research, teaching, and private study purposes.

Any substantial or systematic reproduction, redistribution, reselling, loan, sub-licensing, systematic supply, or distribution in any form to anyone is expressly forbidden.

Authors alone are responsible for the contents of their articles. The journal owns the copyright of the articles.

The publisher shall not be liable for any loss, actions, claims, proceedings, demand, or costs or damages whatsoever or howsoever caused arising directly or indirectly in connection with or arising out of the use of the research material.

Teaching GUI-Programming Concepts to Prospective K12 ICT Teachers: MIT App Inventor as an Alternative to Text-Based Languages

Can Mihci, Nesrin Ozdener Donmez

Article Info	Abstract
<p><i>Article History</i></p> <p>Received: 26 November 2016</p> <p>Accepted: 30 June 2017</p> <hr/> <p><i>Keywords</i></p> <p>Programming education GUI-programming Teacher education Visual programming</p>	<p>The purpose of this research is to investigate the short and long-term effects of using GUI-oriented visual Blocks-Based Programming languages (BBL) as a 2nd tier tool when teaching programming to prospective K12 ICT teachers. In a mixed-method approach, the effect on academic success as well as the impact on professional opinions and preferences have been gathered for drawing deeper conclusions. Conclusively, it was understood that visual BBL increased success for students who were previously having a hard time dealing with text-based programming. However, students that were already accustomed to text-based programming have shown failure adapting to the BBL. This has been interpreted as a failure of previous programming courses in terms of enabling learning transfer. Additionally, it has been understood that employing the BBL has not been entirely effective in causing prospective ICT teachers to think that programming is a subject suitable for a considerably younger target audience.</p>

Introduction

Researchers have frequently pointed out that computer programming is a very difficult subject for students (Cornforth, 2014; Fujiwara, Fushida, Tamada, Igaki, & Yoshida, 2012; Lahtinen, Ala-Mutka, & Järvinen, 2005; Proulx, 2000). It has even been mentioned that one out of three students fails the class at entry level programming courses in universities (Bennedsen & Caspersen, 2007) and that programming is one of the most difficult subjects in the undergraduate curriculum (Jenkins, 2002). Meisalo and colleagues also reported that 32% of college learners have dropped out their programming courses, which were offered as distance education and when asked for their reasons, students all pointed out that the subject was simply too difficult (Meisalo, Suhonen, Torvinen, & Sutinen, 2002). Kinnunen and Malmi report that there is a global trend in universities, which indicates that 20% to 40% of students either drop out at these courses or even quit school altogether as far as IT related programs are concerned (Kinnunen & Malmi, 2008). Another multi-national and multi-organizational study carried out by McCracken and colleagues state that the average success level of students in entry-level programming courses in universities is fairly low (McCracken et al., 2001). Therefore, many university departments that teach IT subjects are seeking an answer as to how courses that introduce students to programming should be handled (Allison, Orton, & Powell, 2002). The already existing problem of the difficulty of programming courses may get even more complicated when the students are expected to not only learn but also to teach in the future to pupils of very young ages, programming concepts. Such is the case with prospective K12 ICT teachers.

As stated by many researchers so far, there is yet another factor that contributes into entry-level programming courses: the fact that the classroom consists of students that possess prior knowledge and experience in ICT and programming subjects at largely varying degrees (Davis, Carr, Cooke, & White, 2001; Jenkins & Davy, 2002; Rößling & Freisleben, 2000) and that this kind of situation is becoming more and more common with each passing year (Hagan & Markham, 2000). The problem posed by the large gap observed between the experienced and novice students has two dimensions. Due to observing how their peers are struggling, experienced students are prone to perceive the subject as easier than it is, which leads them into a state of neglect towards the course. Novice students, on the other hand, compare themselves with experienced students and tend to lose their motivation due to the feeling of inferiority. Several researchers have come up with ideas to overcome this problem (Sanders & Mueller, 2000; Shackelford & LeBlanc Jr, 1997) and mentioned a need for supportive courses for novice students in settings of IT education. All and all, it can be said that existing knowledge or lack thereof of students may impact their performance in programming courses. Therefore, research conducted for the purpose of examining success in IT courses should perhaps be designed to take into account the knowledge and experience levels of pupils.

Several factors that have an impact on student success in programming education have been highlighted in the current literature. Another important one among these is the choice of programming language taught at the courses. In this context, the question as to which programming languages and which paradigms should be used to introduce students to programming, has been a popular one in the field of computer science education.

Blocks-based visual programming languages and their associated development environments follow -as the name suggests- the “visual” and “component-based” programming concepts. These enable the developer to create programming expressions by combining through “drag and drop” or other GUI actions blocks that visually represent programming constructs, as opposed to using text to form expressions in the conventional “text-based” approach.

Although the original purpose of this approach to programming has been to create easier and more practical means for the end-users to develop software applications (Mohamad et al., 2011) there have been researchers claiming that blocks-based visual languages may be used as educational tools in programming (Navarro-Prieto & Cañas, 2001) and that they may be beneficial especially for novice students (Maloney, Resnick, Rusk, Silverman, & Eastmond, 2010). In fact, the programming blocks approach have been considered to be such a great way to simplify programming that, there even exists research that aims to teach programming concepts to young children by employing physically tangible blocks (Horn & Jacob, 2006; McNerney, 2004; Wang, Zhang, & Wang, 2011; Wyeth & Purchase, 2002). The main advantage to blocks-based visual programming is that textual syntax, which may largely differ from one programming language to the other, has been replaced with universal blocks that interact with each other only in meaningful ways, much like pieces of a puzzle, leaving less room for error. This enables novices to practice the semantics instead of syntax, lets them code in elements that display a higher-level fashion resembling pseudo-code, eliminates the need to “memorize” textual code constructs (which is especially the problem in students that do not have English as their mother tongue), while avoiding typing mistakes and type errors in general (Klassen, 2006). However, it should also be noted that blocks-based visual languages that have been used in education so far have mostly been prepared for a target audience of younger age levels in mind (Kelleher, Pausch, & Kiesler, 2007; Utting, Cooper, Kölling, Maloney, & Resnick, 2010). As such, the idea of using blocks-based visual languages to aid the education of adults in programming has not been encountered often in the literature, at least for a certain period. With the advent of the global learn-to-code movement, however, more and more adults are learning to program using these tools, so that they may teach what they learn to children. This notion is important when considered in the context of prospective K12 ICT teachers.

One of the relatively new ideas for enhancing programming education has been to use smart mobile devices, which have gained immense popularity in such short notice, as target platforms for developing application software (Burd et al., 2012; Fenwick Jr, Kurtz, & Hollingsworth, 2011; Mahmoud & Popowicz, 2010; Riley, 2012). Since, it is entirely possible that the desktop computer, which has made its way into almost every living space in the world during the last decade and has been serving as a tool that motivates people in learning to program, has been around for quite a while and may now be considered “obsolete” and taken for granted by the young population. However, the novelty effect and the undeniable popularity of smart mobile devices may help achieve the with today’s young generation what the desktop computer revolution once achieved in the past.

It can be seen that programming languages that are most commonly used in developing native applications for smart mobile devices (Java, Objective C, Visual C#, etc.) are mid-to-high level, text-based, compiled programming languages with an emphasis on the object-oriented paradigm. However, in addition to the previously mentioned problems imposed upon novices by text-based languages, it has also been claimed that education in the object-oriented paradigm may render things more difficult for novices by increasing cognitive load (Wiedenbeck & Ramalingam, 1999).

In the recent years, a new generation of programming languages and tools that enable end users to develop application software for smart mobile devices has been observed. A special breed of these, which possesses the visual blocks-based approach has also emerged. Examples include GameSalad (Dekhane & Xu, 2012; Kaushik Roy, Rousse, & DeMeritt, 2012), Stencyl (Liu et al., 2014) and MIT App Inventor.

MIT App Inventor, which was opened to public access in 2011, is a tool that is comprised of a visual blocks-based programming language and its cloud-based development environment, which aims to help end-users develop application software for smart mobile devices that run the Android operating system. From the programming education perspective, it stands out among the other blocks-based languages with its features of (a) possibly targeting not only children but adults (b) targeting smart mobile devices, (c) enabling the creation of real-life applications that might actually prove useful to a larger audience (Spertus, Chang, Gestwicki, &

Wolber, 2010; Wolber, 2011). These features led educators to soon discover that App Inventor may genuinely possess educational value, hence leading to many papers proposing the use of App Inventor in programming classes (Gray, Abelson, Wolber, & Friend, 2012; Karakus, Uludag, Guler, Turner, & Ugur, 2012; Krishnendu Roy, 2012; Spertus et al., 2010; Wolber, 2011). These publications emphasized the fact that App Inventor may increase student motivation and its simplicity may benefit novices. Ahmad and Gestwicki (Ahmad & Gestwicki, 2013) even proposed that App Inventor may be suitable for college-level programming courses, whereas Karakus and colleagues (Karakus et al., 2012) claimed that App Inventor may be used in programming classes beyond the first and even with experienced students. However, it seems that although the literature displays papers that suggest App Inventor's use and may have reached a consensus regarding its effectiveness, there might be a lack of practical and/or comparative research that serves to solidify the effectiveness of this tool in different settings.

GUI-First

It is true that today, many suggestions on the use of App Inventor and similar software exists. However, a mostly overlooked fact regarding these suggestions is that engaging novice students with App Inventor and similar tools, which serve to develop GUI-applications for smart mobile devices, may be synonymous with employing the "GUI-first" (Yau & Joy, 2004) approach in programming education. Therefore, recommendations regarding App Inventor might be considered de facto implications for employing a "GUI-first" approach, made consciously or otherwise.

The GUI-first approach suggests that students should initially learn to develop applications that display Graphical User Interfaces, instead of "warming them up" with simple console applications that are typically developed in the structural paradigm. This enables the students to learn from day one about visual GUI components that are "objects" and that "inherit" from certain "classes"; which, in turn, helps them grasp object-orientation concepts early on and then move towards the structural/procedural concepts (Decker & Hirshfield, 1999; Proulx, Raab, & Rasala, 2002). GUI components are able to offer a more advanced interface console applications that are typically no more than "black screens" covered in static text (Martinez, 2011). Working on GUI applications obviously increases engagement and motivation of the student, as opposed to working with console applications (Hadjerrouit, 1998) since, students who work with GUI are more immersed in the idea that they are creating applications that are more practical and relevant to the real world (Gibbons, 1998). However, the same researcher suggests that there are certain disadvantages to the GUI-first approach, owing to the fact that focusing on GUI aspects of a program may prevent the student from focusing on the fundamental skill of algorithm development and structural programming, let alone object orientation concepts. In addition, when creating a GUI, visual design concerns may lead the student to lose focus and time.

MIT App Inventor: Not as an Initial but as a Secondary Tool

Many universities still follow the conventional programming curriculum that suggests the use of text-based languages in usually the imperative, procedural or object-oriented paradigms for the development of desktop console applications, in order to introduce novice students to programming. Considering this, and under the light of the information regarding GUI-first notion, one might think there might be an alternative use for the App Inventor generation of educational tools. That is, using these in the second course that follows up the first "conventional" programming course. This way, students may perhaps have a smoother transition from the console to the GUI and from procedural to object-oriented, thanks to simple GUI designer, visual block components of programming and the motivation brought in by smart mobile devices. This approach may even help reclaim students who have stumbled in the initial course due to the difficulties of using a text-based language and/or losing interest in the course due to finding the console applications irrelevant to the real world. There may also be an added benefit of instilling a broader perspective of programming to students by providing them the means to transfer their initial learning (in console applications, with text based languages) to a different environment (GUI applications, visual languages).

However, current literature tells that text-based languages used in programming courses at the university level are chosen as a result of evaluations that rely largely on the criterion of software industry preferences, due to employment concerns. It is a fact that the industry favors text-based languages over others. Another fact is that visual block-based programming languages are considered to be "too simple" by relatively successful students (Kasurinen, Purmonen, & Nikula, 2008). These may be considered elements that may negatively affect student motivation in a course that uses a visual, blocks-based language. Therefore, factors such as prior knowledge or

experience in programming may further influence student performance in a class that uses visual languages and this fact needs to be considered in research efforts. Research also needs to focus on reasons for adoption/rejection of a certain blocks-based language by students.

Programming Education for Prospective ICT Teachers

The rising trend of m-Learning translates into the increased use of smart mobile devices and a greater demand for software applications that run on these platforms. In order to prepare a workforce of K12 ICT teachers and to answer an increasing demand towards educational software developers that carry a certain level of expertise, Turkey has launched in 1998 as part of an initiative for nationwide restructuring of the Education Faculties in the Republic of Turkey the Computer Education and Instructional Technology (CEIT) undergraduate departments. Their founding rationales have been explained as follows: (1) to train ICT teachers that have possess fundamental occupational knowledge and skills, who will teach at K12 level (2) to train Computer-Assisted Teaching experts who possess the skills for designing, developing, implementing and evaluating educational materials and software applications that are suitable for the pedagogic levels of their target audiences and the national curricula (Gerek & Kurt, 2010).

Under the light of this information, it can be said that introducing undergraduate prospective teachers and educational technologists, such as those enrolled in the Turkish CEIT program with tools such as App Inventor may bring about certain benefits - in addition to the obvious benefits of higher performance in programming courses. First of all, this sort of end-user programming interface may help educational software developers to more easily and rapidly design and develop educational software for use in m-Learning. Secondly, App Inventor may be presented to prospective ICT teachers as a classroom tool for use in their future career, where they might most certainly be required to teach young children how to code. However, the validity of this claim needs to be checked and the possible outcomes of the implementation of this tool in such a way should be examined.

Purpose

The purpose of this research is to investigate the short and long-term effects of using GUI-oriented visual Blocks-Based Programming languages (BBL) as a 2nd tier tool when teaching programming to prospective K12 ICT teachers. The said effects are within the scope of a) academic success and; b) professional opinions and preferences in the context of programming education.

Considering two groups of teacher candidates who have initially been taught at 1st tier course introductory programming using a text-based language and by developing desktop console applications, the following hypotheses have been formulated:

1. Compared in terms of academic success in a 2nd tier course dealing with GUI-programming skills, the group that was taught with a visual, blocks-based programming language (BBL) is expected to be more successful than the group that was taught the same concepts with a text-based language (TBL).
2. There is an influence of the factor of being “experienced” or “novice” upon the effect of using BBL instead of TBL in GUI-programming courses at university level

Considering two groups of teacher candidates, one of which have been taught programming using both the TBL and the BBL and the other having been taught programming only with the TBL; the following research questions have been asked:

1. Will the opinions prospective ICT teachers as to what age programming education should start in children show any difference between the groups?
2. How will the preference of programming language differ between two groups of prospective ICT teachers when given a microteaching assignment seeking to teach 8th-grade children a reference topic in programming?

Method

Research Model

Designed as a mixed method study, this research is comprised of two phases. In the first phase, a quasi-experimental pattern has been followed and the quantitative data obtained in this manner has been analyzed statistically. During the second phase, a case study approach has been followed, qualitative data have been gathered from groups which have then been analyzed by coding text content.

Study Group

As detailed in Table 1, the study groups for the research have been 2nd and 3rd-year students who are undergoing undergraduate education at the Department of Computer Education and Instructional Technologies of a Turkish university's faculty of education.

During Phase 1, which comes first chronologically, two different classes that took during Period 2 the course Programming Languages 2, which covers the subject of GUI-programming, were examined.

During Phase 2, students from both classes in Phase 1 were considered altogether as Experimental Group 2. These students were compared with another group of students, namely Control Group 2, who took the Programming Languages 1 and 2 courses under identical conditions in the previous academic year (Period 1).

Table 1. Study groups

Group	Period	# of students	Group	Period	# of students
Experimental Group 1 (Phase 1)	2	54	Experimental Group 2 (Phase 2)	2	101
Control Group 1 (Phase 1)	2	47			
			Control Group 2 (Phase 2)	1	76

Data Collection Instruments

The following items have been used as data collection instruments throughout the study:

Console Application Development Skills Test: Composed of both short answer and multiple choice questions, this test sought to measure knowledge and skill in the following procedural programming concepts within the context of desktop console application development using the text-based C# programming language as part of the .NET software development framework: a) Data Types, b) Assigning Variables c) Logical Operators d) Decision Structures e) Loop Structures f) Functions, g) Arrays. It was scored over 100 points and data obtained from a separate group has shown that the Cronbach's Alpha internal consistency coefficient for this test was 0,81. The test was used for the purpose of determining whether or not the groups in Phase 1 be considered equivalent to one another in terms of procedural programming skills learned at the previous semester's Programming Languages 1 course, by examining if there is a statistically significant difference between their average scores statistically. A sample question from this test is as shown in Figure 2

- Write the output of the program code given below.

```

static void Main(string[] args){
    int i, a=3;
    for (i=1; i<10; i++){
        if(i % a == 0){
            Console.WriteLine("{0} ", i);
        }
    }
}

```

Figure 2. Sample question from Console Application Development Skills Test

This test also included a single question that asked whether or not a student received GUI-programming education prior to their university education, be it as part of high school education or in any other way such as through another educational program, distance or otherwise. Upon examination of data, the students who declared they never took GUI-programming courses prior to university education were considered as previously “Uneducated” in terms of GUI-programming.

GUI-Programming Skills Test: This test sought to find out the level of skill in GUI-programming of students who declared in the Console Application Development Skills Test that they received GUI-programming education prior to university education. The Cronbach’s Alpha internal consistency coefficient for this multiple-choice test was found as 0,94. Only those students who scored above-average in this test were considered as previously “Educated” in terms of GUI-programming. The rest were still considered uneducated. The test was used for two purposes: (1) To determine whether or not the study groups of Phase 1 could be considered equivalent to one another in terms of average GUI-programming knowledge from before university education (2) to help determine the factor groups “experienced” and “novice” in terms of overall programming education.

GUI-Programming Skills Post-Test: This exam was administered to students from both Experimental Group 1 and Control Group 1 at the end of 6 weeks of GUI-programming education. The exam was carried out in the computer laboratory and the students were expected to carry out programming tasks with their computers by using the respective programming language they received the 6-week education with (BBL for Experimental Group 1, TBL for Control Group 1). The purpose has been to measure grasp of students over GUI-programming concepts. Special caution was taken in designing the exam so that the questions were platform-and-language agnostic. This means that consideration was taken to ensure that each question was addressing only specific universal learning objectives in terms of GUI-programming. A sample question from this exam is given in Figure 3. Note that the question explanatory image is prepared in a platform-agnostic way, i.e. not favoring desktop or mobile interfaces over one another.

“The squirrel, the cat, the mouse and the tortoise have been trapped inside a musty cellar! They find the door open, however, and must make a run for their freedom. With your help, of course. Please follow the steps to develop this little game. (The video displaying the finished version of the project has been shared at the Desktop folder for reference, along with sprites necessary to build it)

a) *Make sure that the title bar features the name of the game “Büyük Kaçış”*

b) *Make sure that game area containing the character sprites and the door is of an appropriate component type with a size of 250 by 200 pixels*

...

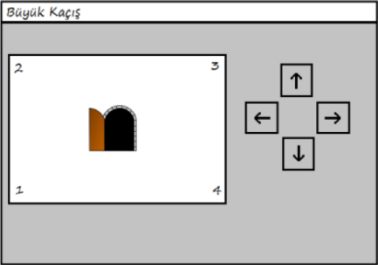


Figure 3. Sample question from the practical exam

Microteaching Assignment: This task was assigned to both Experimental Group 2 and Control Group 2 as part of Phase 2. The students were asked to prepare a lesson plan report and shoot a 20-minute long microteaching video of themselves accordingly, for the objective of teaching a reference subject of “Loop Structures in Programming” to a reference age group of 7th Grade students. This assignment was given to students 1 year after the Programming Languages 2 course, where Experimental Group 2 students have received GUI-programming education using both BBL and TBL and the Control Group 2 students received the same education with only TBL. The purpose has been to find out whether or not the student’s preferred programming language in designing the micro-teaching would differ based on having officially been educated with the BBL.

Procedural Information

Phase 1: Two groups of students, whose members were fixed prior to the beginning of Phase 1 have been determined as Experimental Group 1 and Control Group 1 through random selection. As per the posttest control group quasi-experimental design, two groups that are statistically non-inferior to one another in programming skills have been subjected to GUI programming education for 6 weeks, with the Experimental Group 1

receiving the education over the BBL and the Control Group 1 receiving it over the TBL. At the end of the 6-week education period, the same Practical Exam 1, which measures GUI-programming academic success, was administered to both groups and the average scores of groups have been analyzed statistically and in accordance with various factors. At the end of the 6-week education, the language in groups have been switched and another 6-week long GUI programming education was given to the students, this time with the other language. This means that, by the end of the semester, both groups have received 12 weeks of GUI programming education with both the BBL and the TBL, with 6 weeks spent studying with each language.

The course was offered by the same instructor to both groups using the same classroom facilities, with 3 hours of theoretical and 2 hours of applied course hours each week. Both classes were taught each taught the same GUI programming concepts each week and again, the same learning objectives were followed each week. This means that, in a given week, both classes focused on the same GUI programming project that was adapted to the BBL and TBL environments.

Classroom projects in both languages included but were not limited to TBL and BBL versions of games and utility applications such as Bingo Game, Dictionary, Word Racer Game, and a few CRUD applications. Special consideration was given to make TBL and BBL versions of projects as identical to one another as possible in terms of visuals, algorithms and overall functionality. A sample classroom project of “Bingo” with its versions made in both App Inventor and .NET C# have been given in Figure 1.

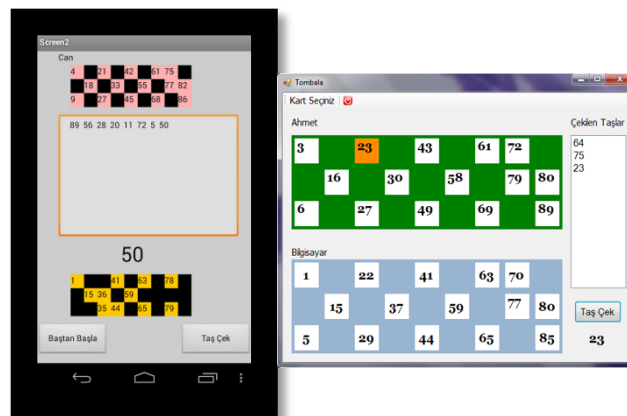


Figure 1. The two versions, namely (a) App inventor and (b) .NET C#

Both classes had received in the previous semester the course Programming Languages 1, which covers the subject of procedural programming by developing desktop console applications using the text-based programming language C# within the .NET software development framework. Both of the classes had students that took programming courses prior to university education, as well as students that did not. For the purpose of Phase 1, one of the classes was randomly selected as the Experimental Group 1, with the other being designated as Control Group 1. The Experimental Group 1 received GUI-programming education with the BBL for the entire first 6 weeks, whereas Control Group 1 received education with the TBL. At the end of 6th week, data has been collected and Phase 1 of the study was finished. From weeks 6 to 12, the groups switched the programming languages with which they took the course, i.e. the Experimental Group 1 started using the TBL for the remainder of the course whereas the Control Group 1 started using the BBL from the 7th week and on.

Phase 2: The second phase of the research begins one year after the first and deals with the long term effects of having studied GUI programming concepts using the BBL. Therefore, the Experimental Group 1 and Control Group 1 have been combined into a single group, namely “Experimental Group 2” exactly one year after the education received during Phase 1, and qualitative data regarding their professional views of GUI programming education have been collected via surveys and video footage.

The respective “Control Group 2” for this phase of research is another group of students who have undergone the 12-week GUI-programming education with the same instructor and the same TBL as the Experimental Group 2, exactly one year before Phase 1 commenced. The same qualitative data as Experimental Group 2 have been gathered from this group, in the same manner, exactly 1 year after the 12-week course. It should be noted that this group has received but a 15-minute demonstration of the BBL during their GUI-programming course,

which showcased its features, and this was the only official introduction and BBL was not covered as a subject throughout the 12-week course.

Qualitative data from these two groups have been coded and compared as per the relational survey method of research in order to determine the effects of simply having heard of and having officially received GUI-programming education with the BBL, upon the professional views of prospective K12 ICT teachers

Findings

Phase 1 Pre-Intervention Findings

In order to determine whether or not the study groups in Phase 1 can be initially considered equivalent and therefore comparable with one another in terms of relevant programming skills, statistical non-inferiority tests that measured prior knowledge in the relevant programming skills (namely, Prior Knowledge Tests 1 and 2) have been carried out.

It was understood that there was no statistically meaningful difference between the average Prior Knowledge Test 1 scores of groups ($p=.93$). The groups have therefore been considered initially equivalent and comparable to one another in terms of procedural programming skills acquired at the Programming Languages 1 course.

It was understood that there was no statistically significant difference between Prior Knowledge Test 2 results of groups ($U=1212,00$ $p>.05$). The groups have therefore been assumed to be initially equivalent and comparable to one another in terms of GUI-programming skills prior to the intervention.

After they have been allocated to factor subgroups depending on a) their success in the previous Programming Languages 1 course and b) level of education in GUI programming subjects prior to university, after being filtered by a gap that delineated the difference between low achievers and high achievers in each respective test. Final appearance of factor subgroups for each group has been as shown in Table 2.

In accordance with this distribution, the combinations of “uneducated” (in terms of GUI programming) and “unsuccessful” (in terms of a previous programming course in university) factor groups have been designated as subgroup of “Novices” in each group ($N = 12$ for the Experimental Group 1 and $N = 12$ for Control Group 1); while combinations of “educated” and “successful” in each main group have been designated as “Experienced” ($N = 11$ for the Experimental Group 1 and $N = 10$ for Control Group 1).

Table 2. Final appearance of factor subgroups

Experimental Group 1		GUI Prog. Ed. Prior to University	
		Uneducated	Educated
Success in PL1 Course	Unsuccessful	[12]	3
	Successful	4	[11]
Control Group 1		GUI Prog. Ed. Prior to University	
		Uneducated	Educated
Success in PL1 Course	Unsuccessful	[12]	6
	Successful	3	[10]

Findings Pertaining to Hypotheses on Academic Success

For the testing of Hypothesis 1.1, the scores obtained by the groups from the Practical Exam have been compared in an Independent Samples t-test as shown in Table 3.

Table 3. Results of the independent samples t-test

Groups	N	\bar{x}	S	t-test		
				df	t	p
Experimental Group 1 (App Inventor)	54	37,96	18,23	99	,711	.48
Control Group 1 (.NET, C#)	47	41,06	25,38			

As seen by the results of the t-test, the programming language that is used has no significant effect upon success in a practical exam, which measures GUI programming skills ($t(99) = .71, p > .05$).

The factor combination of Experienced/Novice has been formed with the intersections of the factors of success in previous Programming Language 1 course and GUI-programming education prior to university. Hypothesis 1.4, which states that there is an interaction between the effect of this factor combination and the programming language used upon Practical Exam scores, has been tested with a 2x2 factorial analysis of variance statistical analysis. Descriptive statistics for this test and the results of the Shapiro-Wilk test of normal distribution have been given in Table 4.

Table 4. Descriptive statistics and tests for normal distribution

Factor Subgroups	n	\bar{x}	S	Shapiro-Wilk		
				Statistic	df	p
Experimental Group 1, Experienced	11	46,36	16,74	,94	11	,58
Experimental Group 1, Novice	12	34,58	19,70	,94	12	,53
Control Group 1, Experienced	10	67,00	23,11	,92	10	,37
Control Group 1, Novice	12	20,00	11,87	,87	12	,06

Normally distributed data with a homogeneous distribution of error variances (Levene Test of Equality in Variances, $p > .05$) have met the requirements for a test of 2x2 factorial analysis of variance, the results of which have been given in Table 5.

Table 5. Interaction between the effects of the programming language and the experience

Factors	Source of Variance	Type III Sum of Squares	df	Mean Squares	F	p
Programming Language X Experience	Prog. Lang	102,46	1	102,46	,31	,57
	Experience	9662,63	1	9662,63	29,48	,00
	PxE	3468,99	1	3468,99	10,58	,00
	Error	13437,46	41	327,74		
	Total	26297,77	44			

The results have shown that there is a statistically meaningful two-way interaction between the effects of the “Experienced/Novice” dichotomous factor combination and the programming language used in the Programming Languages 2 course, upon the Practical Exam scores ($F(1,44) = 10,58, p = .00$). The “Experienced/Novice” factor combination also has a significant simple main effect upon academic success in GUI-programming, as measured by the Practical Exam ($F(1,44) = 29,48, p = .00$). The profile graph for this test has been given in Figure 4.

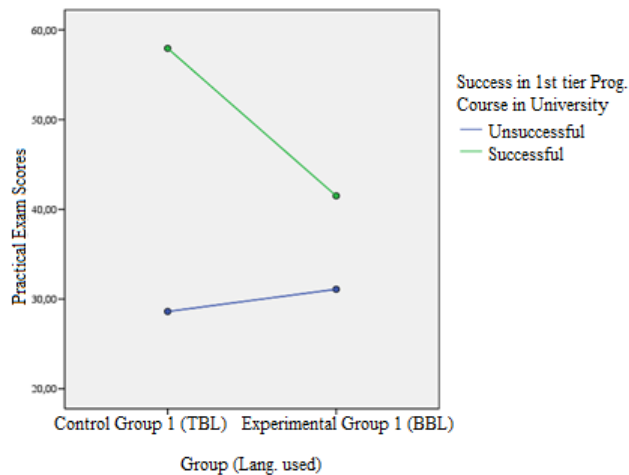


Figure 4. Interaction between P X E

The graph in Figure 4 may be interpreted as follows: As far as the Programming Languages 2 (PL2) course, which aims to cover GUI-programming concepts is concerned, the students labeled as “experienced”, who have taken courses in GUI-programming skills prior to university and who have also shown success in the previous Programming Languages 1 course; show significantly greater academic success when they used the TBL instead of the BBL in the PL2 course. Whereas, students labeled as “novice”, who never knew about GUI-programming concepts before this course and who have performed poorly in the previous Programming Languages 1 course; show significantly greater academic success when they used the BBL instead of the TBL in the PL2 course. This situation has been further analyzed with an Independent Samples t-test, the results of which have been shown in Table 6.

Table 6. Results of the independent samples t-tests

Groups	n	\bar{x}	ss	t-test		
				sd	t	p
Experimental Group 1 (BBL) Novice	12	34,58	19,70	22	-2,20	.039
Control Group 1 (TBL) Novice	12	20,00	11,87			
Groups	n	\bar{x}	ss	t-test		
Experimental Group 1 (BBL) Experienced	10	67,00	23,11	19	2,35	.03
Control Group 1 (TBL) Experienced	11	46,36	16,74			

It can be seen that, as far as average scores in the Practical Exam are concerned, there are statistically significant differences between both novice ($t(22) = -2.20$, $p < .05$) and experienced ($t(19) = 2.35$, $p < .05$) students based on the programming language they have used in the GUI-programming course, with novices being more successful using the BBL and experienced students more successful in the TBL.

Findings Pertaining to Research Questions

Certain open-ended questions have been asked to students, who are also prospective K12 computer science teachers, exactly one year after they have completed their programming courses at university, which they receive in the 2nd year of their undergraduate program. At this phase, the Experimental Group 2 consisted of students who were taught programming courses with both the TBL and the BBL. The Control Group 2 consisted of students who received their university programming courses entirely with the TBL, and the BBL has been introduced to them briefly, for 15 minutes, as an educational tool that can be used for teaching children to program. Student answers for the questions have been treated as qualitative data and coded into content categories using the NVivo qualitative data analysis software suite.

The prospective computer science teachers in the Experimental Group 2, who have been taught programming with both the BBL and the TBL; and Control Group 2, who have been taught programming with only the TBL have been asked an open-ended question for their opinions on whether or not elementary school children should be taught programming. Student answers have been coded into categories, and the distribution of these have been shown in a graph in Figure 5.

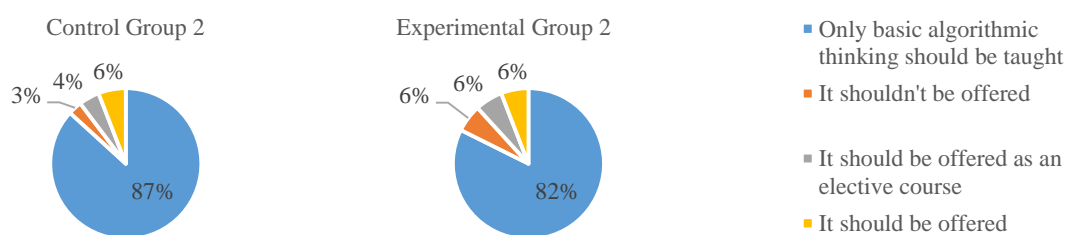


Figure 5. Should elementary school children be taught programming?

Other themes that were established while coding student responses to this question are as follows:

1. The implication of having the subject of programming introduced to children at an early age such as elementary school could be “useful/beneficial” for the child considering how it could “help direct” her for the “future” and provide her with ease in choosing a job. (Experimental Group 2: n=9, 34.62%, Control Group 2: n=8, 38.10%) :
2. The implication that building GUI applications, as opposed to console applications, may prove to be particularly “interesting” and “fun” for elementary students (Experimental Group 2: n=4, 19.23%, Control Group 2: n=2, 9.52%):
3. The implication that programming education may be beneficial for the student in alternative ways, such as improving the development of intelligence (Experimental Group 2: n=8, %30.77, Control Group 2: n=9, %42.86)
4. The implication that, considering the conditions of today, it may be more appropriate to introduce children to programming at earlier ages (Experimental Group 2: n=4, %15.38, Control Group 2: n=2, %9.52).

Prospective computer science teachers have been asked an open-ended question for their opinions on the age for starting programming education. Answers have been coded into categories, and the distribution of these have been shown in a graph in Figure 6

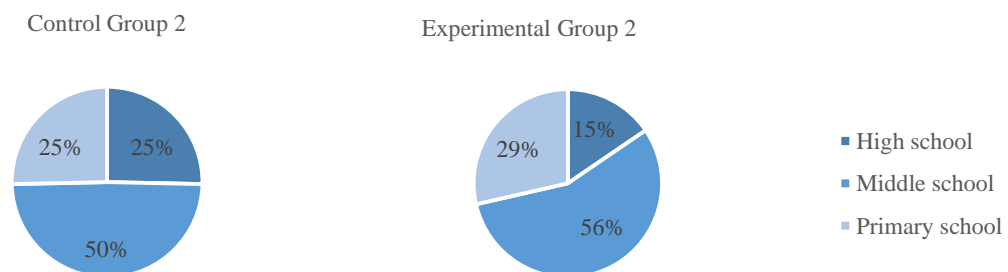


Figure 6. The effect of the languages used in programming course at university education upon the views of prospective CS teachers on the age level programming education should start, comparative pie graphs

Upon inspection of the graph, it was understood that students who were educated with both the TBL and the BBL displayed a slight tendency in their answers towards earlier ages for beginning computer programming education (29% primary, 56% middle school), as compared to students who were educated solely with the TBL (25% primary, 50% middle school). The Control Group 2 students thought programming should be a high school level subject (25%) more often than the Experimental Group 2 (15%). Therefore, it could be said that having received education with the BBL may have caused students to think programming education should start at earlier ages.

Each student in both the Experimental Group 2 and Control Group 2 have been asked to shoot one 20-minute long video of themselves, in which they were teaching to an imaginary audience of 8th-grade students with the necessary prior knowledge and pedagogical availability the reference subject of “FOR loops in programming”. The students were told that they are free to choose any programming language and environment they want, in teaching the subject. The videos have then been examined to find out which programming languages the prospective CS teachers have chosen, depending on the group they belonged to. The graph for the distribution of programming languages by student choice for Experimental Group 2 and Control Group 2 have been shown in Figure 7.

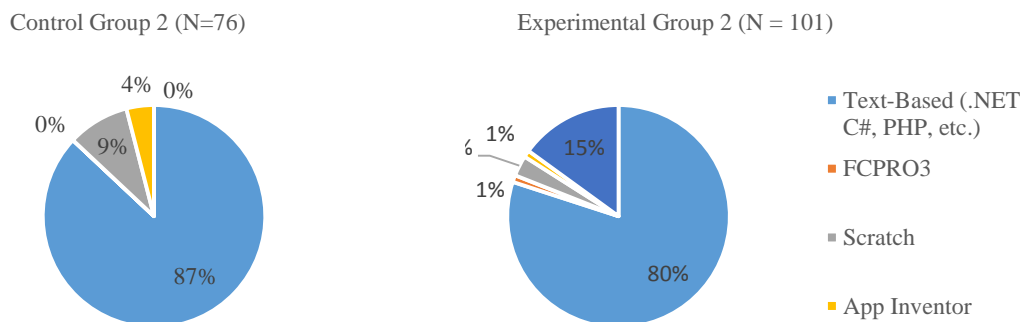


Figure 7. Languages/tools used in micro-teaching videos

It was seen that out of all the students in the Experimental Group 2 ($n = 101$), 80% have chosen to use a text-based language in their micro-teaching videos; whereas 5% have chosen visual languages. The 5% portion is comprised of students that used the software flowchart tool FCPRO3 for the task (1%), in addition to the 3% that used MIT Scratch and 1% that used the BBL used in this study, namely “MIT App Inventor”. 15% of the students in this group did not use a specific language or environment in their videos.

As for the students in Control Group 2 ($n = 76$), a portion of 87% has chosen to teach the reference subject using a text-based language. The remaining 13% have chosen to use visual languages for the task, with 9% of this being comprised of MIT Scratch and 4% being MIT App Inventor.

Discussion

This study is showing the findings of a comparative case study wherein undergraduate university students at a Department of Computer Education and Instructional Technologies, who come from different educational backgrounds and who are also prospective CS teachers for K12 schools, have been taught GUI-programming concepts in a 2nd tier Programming course using different programming languages in the two compared cases. Whereas one case was receiving the course using a text-based programming language for developing GUI applications targeting a desktop environment, the other case was receiving the course using a blocks-based visual programming language for developing GUI applications targeting smart mobile devices. Learning outputs of the course have been considered for ensuring that the classroom activities in both cases -including but not limited to sample projects and homework have been designed to be identical, with the only difference being the programming language and environment used. The purpose has been to investigate the effect of using a different, block based visual programming language for developing GUI applications targeting smart mobile devices, upon the academic success of students. The different educational backgrounds and success in previous programming classes have been taken into consideration in doing so. It was also questioned whether using the blocks-based language would influence the professional opinions of the prospective CS teachers.

Initial findings have shown that using the blocks-based language instead of the text-based language did not create a significant impact upon student academic success in learning GUI-programming concepts. However; considering the fact that some of the students came from vocational high schools, where they have received extensive programming courses (and possibly gained knowledge in GUI-programming) and that the research has been carried out in a second programming course in the undergraduate program, these factors have been included in statistical analyses.

The separate situations have been combined to form the overall “experienced” and “novice” groups of students. The experienced group consisted of students that both had a high school background in GUI-programming and text-based languages, and have performed well in the first tier programming course at the University. The novice group consisted of students that had no high school experience in programming and performed poorly at the first tier course.

In this case, it was seen that novice students performed significantly better using the blocks-based language and experienced students performed significantly better using the text-based language. This phenomenon could be an indicator of students who get their first introduction to programming at later years may be performing poorly due to learning difficulties that are associated with features specific to text-based programming, such as having to memorize syntax. At this point, blocks based languages seem to assume the role of savior for the so-called

“novice” students. On the other hand, another interesting phenomenon, which involved the so-called “experienced” students performing significantly worse using the blocks-based language, was observed.

Academic literature suggests that using a blocks-based visual programming language for mobile application development, namely the MIT App Inventor, for introductory programming education may be beneficial for students particularly due to a potential increase in student motivation based on developing mobile apps (Ahmad & Gestwicki, 2013). There have been interesting claims regarding the potential of App Inventor for use in university-level programming courses, stating that this language could be an educational tool that can be used even by CS-majors and even in courses beyond CS0 (Karakus et al., 2012). These researchers have stated that App Inventor has the potential to keep alive student interest in programming. However, findings of this here study, where all participants have taken at least one programming course in the past, are partly inconsistent with claims of Karakuş et al. regarding the use of App Inventor beyond CS0. The findings of this study, which seeks to find the effect of using App Inventor in a more advanced programming course with students who have already been introduced to programming with conventional text-based languages; point to the notion that the audience that may benefit from the use of App Inventor in this manner is limited to students who were either introduced to computing at later age and had performance problems programming in the text-based language. Moreover, although it was expected for students who had high school experience in GUI programming and who were found to be proficient in procedural programming with text-based languages in a first tier university programming course to be at least as successful as using their favorite text-based language when using MIT App Inventor; this has not been the case. The “experienced” students have shown difficulty transferring their existing knowledge of programming to the blocks-based environment of App Inventor.

Although it was expected of App Inventor to increase student success through motivation (Ahmad & Gestwicki, 2013), there may be a reason for more experienced students to show disinterest in this educational tool. The fact that the text-based language used in this study as an alternative for App Inventor, .NET C#, is a professional programming language that is a highly popular choice in many industries, with the possibility of providing employment opportunities for its user. Knowing this fact may have acted as a motivator for the experienced students in studying harder and learning this language well. Conversely, knowing the fact that App Inventor is the name of a non-profit project mainly used for developing personal and mostly amateur mobile applications and not a requirement in employment ads, may have deterred the motivation of the experienced student group. This situation has been referred to in a previous study, which stated that the popularity of App Inventor may be enhanced by using this language as a bridging tool for moving on to the text-based Java programming language, which is a language that is extremely relevant in the professional world of programming and which is actually the language that App Inventor hides behind its visual blocks (Soares, 2014). Therefore, it may be true that using this tool with a promise of more easily getting introduced to a harder to grasp, yet more industrially relevant language might be a way to increase student motivation in learning scenarios that make use of this tool. This way, an App Inventor – Java hybrid educational model (Soares, 2014) can be established and this, in turn, would be consistent with the recommendation of Jenkins, who stated that considering the professional value of an introductory programming language is almost as important as considering its pedagogical value (Jenkins, 2002, 2004).

However, all the issues discussed herein still fail to provide an answer as to why the “experienced” group of students have failed to perform at least as successful as they did with the text-based language when they used App Inventor. At this point, it may be right to mention the effect of using text-based languages with complicated syntax upon the quality of programming education. Using a low-level language with a relatively complicated syntax during high school programming education makes it more difficult for students to grasp the fundamental concepts of programming (Grandell, Peltomäki, Back, & Salakoski, 2006) and causes them to “memorize” syntactic rules instead of inherently understanding them. This situation is not unlike the notion of cargo-cult programming. When the features that an introductory programming language should possess (McIver & Conway, 1996) are examined, it can be seen that most of today’s highly popular object-oriented programming languages, including .NET C# used in this study, can be considered to have complicated syntax yet still used often in introductory programming education. This may be the reason why the “experienced” students, who have mostly been educated in the text-based C# language during their vocational high school years, in addition to the first-tier programming course in university, have failed to transfer their existing knowledge of programming into the blocks-based environment. This situation conforms with reported problems that novices face in learning programming (Winslow, 1996), and may call for a pedagogical restructuring of programming courses at the K12 level.

More so, it has been stated that as programming students increase their levels of knowledge, they tend to perceive their programming knowledge and capabilities to be higher than they actually are, which leads them to

underestimate the difficulty of course content and experiencing only a mere illusion of competency (Milne & Rowe, 2002). Block-programming, therefore, may have been perceived as “too easy” by the experienced students, which caused them to lose interest in the important details and eventually perform poorly.

Another purpose of the research has been to determine whether receiving programming education that uses the blocks-based language of App Inventor as an educational tool would influence the views on programming education of CEIT students, who are both prospective CS teachers and future educational technologists.

In this context, open-ended questions regarding programming education at the K12 level have been asked. The answers have shown that receiving programming education with App Inventor has caused a slight downward shift in students’ opinion categories, indicating they thought programming education may start a bit earlier. Another vaguely present trend in the group of App Inventor students has been towards the belief of using “GUI development” as a tool in K12 education may make programming more fun and interesting for students.

Lastly, micro-teaching videos of students have been analyzed to see which programming languages the prospective teachers would prefer for teaching the reference subject of “FOR loops in programming” to 8th-grade students. It was observed that both groups have largely preferred using text-based languages for the task. There has also been a drop in the already low rate of students that preferred using App Inventor, among those who received programming education with it.

This situation bears inconsistency with findings of similar studies. For example, a group of researchers has investigated how prospective CS teachers perceive the professional value of the Scratch visual language for programming education at the K12 level (Fesakis & Serafeim, 2009). As a result of this study, it was understood by polling students that they perceived Scratch as a valuable educational tool. Similar results have been found in a research that investigated teacher education in Korea (Choi, 2012). However, there may be a need for interpreting the existing inconsistency based on the precise similarities and differences that App Inventor has with Scratch. On the other hand, it should be noted that these existing studies do not make use of “hands-on” approaches by prospective teachers such as micro-teaching videos and more importantly, that the blocks-based language has not been contested by an alternative, text-based language.

Conclusion and Suggestions

It should not be forgotten that employing App Inventor or similar blocks-based visual programming languages for introducing students to programming means the de facto acceptance of following a “GUI-first” paradigm and that this should always be an informed decision.

In case a text-based language is chosen for introducing novices to programming, it is imperative to prefer a language with a simpler syntax over complicated ones. It is only this way that the novices may focus on the deeper fundamental and universal concepts of programming that are hidden behind the superficial, syntactical details unique to the language itself.

Offering courses with the MIT App Inventor and similar tools at later stages of undergraduate programming education may be beneficial, provided it is used with students who have struggled with more complicated, text-based languages. However, offering courses with these tools for students that have somehow been successfully introduced to programming with text-based languages may cause a drop in student success due to adaptation problems or lack of motivation. The former may indicate a failure in fostering a generally applicable understanding of programming in students, whereas the latter may be alleviated by using the simpler, blocks-based languages as bridging tools for passing over to the more advanced and professionally relevant text-based languages.

This study has failed to uncover a striking effect of having been educated with the blocks-based tool for mobile app development upon the professional perceptions of prospective CS teachers upon programming education. The fact that this situation contradicts with similar studies found in the literature may be due to the fact that the aforementioned studies being based on “self-reported” data that do not have a purpose of comparison with text-based languages. This could mean that studies that focus on the professional perceptions of prospective teachers should be enhanced with hands-on data collection instruments and/or make use of a comparative approach.

Acknowledgements

The study has been funded by Marmara University Research Projects Board as research project number EGT-C-YLP-150513-0211.

References

- Ahmad, K., & Gestwicki, P. (2013). *Studio-based learning and app inventor for android in an introductory CS course for non-majors*. Paper presented at the Proceeding of the 44th ACM technical symposium on Computer science education.
- Allison, I. K., Orton, P., & Powell, H. (2002). *A virtual learning environment for introductory programming*. Paper presented at the Proceedings of the 3rd Conference of the LTSN-ICS.
- Bennedsen, J., & Caspersen, M. E. (2007). Failure rates in introductory programming. *ACM SIGCSE Bulletin*, 39(2), 32-36.
- Burd, B., Barros, J. P., Johnson, C., Kurkovsky, S., Rosenbloom, A., & Tillman, N. (2012). *Educating for mobile computing: addressing the new challenges*. Paper presented at the Proceedings of the final reports on Innovation and technology in computer science education 2012 working groups.
- Choi, H. (2012). *Learners' reflections on computer programming using Scratch: Korean primary pre-service teachers' perspective*. Paper presented at the ICoME 2012 (International Conference on Media in Education), Beijing, China.
- Cornforth, D. (2014). *Teaching mobile apps for Windows devices using TouchDevelop*. Paper presented at the Proceedings of the Sixteenth Australasian Computing Education Conference-Volume 148.
- Davis, H., Carr, L., Cooke, E., & White, S. (2001). Managing Diversity: Experiences teaching programming principles.
- Decker, R., & Hirshfield, S. (1999). *Programming Java: An Introduction to Programming Using Java*: Brooks/Cole Publishing Co.
- Dekhane, S., & Xu, X. (2012). Engaging students in computing using GameSalad: a pilot study. *J. Comput. Sci. Coll.*, 28(2), 117-123.
- Fenwick Jr, J. B., Kurtz, B. L., & Hollingsworth, J. (2011). *Teaching mobile computing and developing software to support computer science education*. Paper presented at the Proceedings of the 42nd ACM technical symposium on Computer science education.
- Fesakis, G., & Serafeim, K. (2009). *Influence of the familiarization with Scratch on future teachers' opinions and attitudes about programming and ICT in education*. Paper presented at the ACM SIGCSE Bulletin.
- Fujiwara, K., Fushida, K., Tamada, H., Igaki, H., & Yoshida, N. (2012). *Why Novice Programmers Fall into a Pitfall?: Coding Pattern Analysis in Programming Exercise*. Paper presented at the Empirical Software Engineering in Practice (IWESEP), 2012 Fourth International Workshop on.
- Gerek, S., & Kurt, A. A. (2010). Bilgisayar ve Öğretim Teknolojileri Eğitimi Bölümlerinde Ekonomi Okuryazarlığına İlişkin Göstergeler. *University of Gaziantep Journal of Social Sciences*, 9(1).
- Gibbons, J. (1998). Structured programming in Java. *SIGPLAN notices*, 33(4).
- Grandell, L., Peltomäki, M., Back, R.-J., & Salakoski, T. (2006). *Why complicate things?: introducing programming in high school using Python*. Paper presented at the Proceedings of the 8th Australasian Conference on Computing Education-Volume 52.
- Gray, J., Abelson, H., Wolber, D., & Friend, M. (2012). *Teaching CS principles with app inventor*. Paper presented at the Proceedings of the 50th Annual Southeast Regional Conference.
- Hadjerrouit, S. (1998). Java as first programming language: a critical evaluation. *SIGCSE Bull.*, 30(2), 43-47. doi:10.1145/292422.292440
- Hagan, D., & Markham, S. (2000). *Does it help to have some programming experience before beginning a computing degree program?* Paper presented at the ACM SIGCSE Bulletin.
- Horn, M. S., & Jacob, R. J. (2006). *Tangible programming in the classroom: a practical approach*. Paper presented at the CHI'06 extended abstracts on Human factors in computing systems.
- Jenkins, T. (2002). *On the difficulty of learning to program*. Paper presented at the Proceedings of the 3rd Annual Conference of the LTSN Centre for Information and Computer Sciences.
- Jenkins, T. (2004). The First Language-A Case for Python? *Innovation in Teaching and Learning in Information and Computer Sciences*, 3(2).
- Jenkins, T., & Davy, J. (2002). Diversity and motivation in introductory programming. *Innovation in Teaching and Learning in Information and Computer Sciences*, 1(1).
- Karakus, M., Uludag, S., Guler, E., Turner, S. W., & Ugur, A. (2012). *Teaching computing and programming fundamentals via App Inventor for Android*. Paper presented at the Information Technology Based Higher Education and Training (ITHET), 2012 International Conference on.

- Kasurinen, J., Purmonen, M., & Nikula, U. (2008). *A study of visualization in introductory programming*. Paper presented at the 20th Annual Psychology of Programming Interest Group Conference, PPIG.
- Kelleher, C., Pausch, R., & Kiesler, S. (2007). *Storytelling Alice motivates middle school girls to learn computer programming*. Paper presented at the Proceedings of the SIGCHI conference on Human factors in computing systems.
- Kinnunen, P., & Malmi, L. (2008). *CS minors in a CSI course*. Paper presented at the Proceedings of the Fourth International Workshop on Computing Education Research.
- Klassen, M. (2006). *Visual approach for teaching programming concepts*. Paper presented at the Proceedings of the 9th International Conference on Engineering Education (ICEE 2006).
- Lahtinen, E., Ala-Mutka, K., & Järvinen, H.-M. (2005). *A study of the difficulties of novice programmers*. Paper presented at the ACM SIGCSE Bulletin.
- Liu, J., Lin, C.-H., Wilson, J., Hemmenway, D., Hasson, E., Barnett, Z., & Xu, Y. (2014). *Making games a snap with Stencyl: a summer computing workshop for K-12 teachers*. Paper presented at the Proceedings of the 45th ACM technical symposium on Computer science education.
- Mahmoud, Q. H., & Popowicz, P. (2010). *A mobile application development approach to teaching introductory programming*. Paper presented at the Frontiers in Education Conference (FIE), 2010 IEEE.
- Maloney, J., Resnick, M., Rusk, N., Silverman, B., & Eastmond, E. (2010). The Scratch Programming Language and Environment. *Trans. Comput. Educ.*, 10(4), 1-15. doi:10.1145/1868358.1868363
- Martinez, W. L. (2011). Graphical user interfaces. *Wiley Interdisciplinary Reviews: Computational Statistics*, 3(2), 119-133. doi:10.1002/wics.150
- McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y. B.-D., . . . Wilusz, T. (2001). A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. *ACM SIGCSE Bulletin*, 33(4), 125-180.
- McIver, L., & Conway, D. (1996). *Seven deadly sins of introductory programming language design*. Paper presented at the Software Engineering: Education and Practice, 1996. Proceedings. International Conference.
- McNerney, T. (2004). From turtles to Tangible Programming Bricks: explorations in physical language design. *Personal and Ubiquitous Computing*, 8(5), 326-337. doi:10.1007/s00779-004-0295-6
- Meisalo, V., Suhonen, J., Torvinen, S., & Sutinen, E. (2002). *Formative evaluation scheme for a web-based course design*. Paper presented at the ACM SIGCSE Bulletin.
- Milne, I., & Rowe, G. (2002). Difficulties in learning and teaching programming—views of students and tutors. *Education and Information Technologies*, 7(1), 55-66.
- Mohamad, S. N. H., Patel, A., Latih, R., Qassim, Q., Na, L., & Tew, Y. (2011). *Block-based programming approach: challenges and benefits*. Paper presented at the Electrical Engineering and Informatics (ICEEI), 2011 International Conference on.
- Navarro-Prieto, R., & Cañas, J. J. (2001). Are visual programming languages better? The role of imagery in program comprehension. *International Journal of Human-Computer Studies*, 54(6), 799-829.
- Proulx, V. K. (2000). *Programming patterns and design patterns in the introductory computer science course*. Paper presented at the ACM SIGCSE Bulletin.
- Proulx, V. K., Raab, J., & Rasala, R. (2002). Objects from the beginning-with GUIs. *ACM SIGCSE Bulletin*, 34(3), 65-69.
- Riley, D. (2012). *Using mobile phone programming to teach Java and advanced programming to computer scientists*. Paper presented at the Proceedings of the 43rd ACM technical symposium on Computer Science Education.
- Roy, K. (2012). *App Inventor for Android: report from a summer camp*. Paper presented at the Proceedings of the 43rd ACM technical symposium on Computer Science Education.
- Roy, K., Rouse, W. C., & DeMeritt, D. B. (2012). *Comparing the mobile novice programming environments: App Inventor for Android vs. GameSalad*. Paper presented at the Frontiers in Education Conference (FIE), 2012.
- Röbbling, G., & Freisleben, B. (2000). *Experiences in using animations in introductory computer science lectures*. Paper presented at the ACM SIGCSE Bulletin.
- Sanders, I., & Mueller, C. (2000). *A fundamentals-based curriculum for first-year computer science*. Paper presented at the ACM SIGCSE Bulletin.
- Shackelford, R. L., & LeBlanc Jr, R. J. (1997). *Introducing computer science fundamentals before programming*. Paper presented at the Frontiers in Education Conference, 1997. 27th Annual Conference. Teaching and Learning in an Era of Change. Proceedings.
- Soares, A. (2014). Reflections on teaching App Inventor for non-beginner programmers: Issues, challenges, and opportunities. *Information Systems Education Journal*, 12(4), 56.

- Spertus, E., Chang, M. L., Gestwicki, P., & Wolber, D. (2010). *Novel approaches to CS 0 with app inventor for android*. Paper presented at the Proceedings of the 41st ACM technical symposium on Computer science education.
- Utting, I., Cooper, S., Kölling, M., Maloney, J., & Resnick, M. (2010). Alice, Greenfoot, and Scratch--a discussion. *ACM Transactions on Computing Education (TOCE)*, 10(4), 17.
- Wang, D., Zhang, C., & Wang, H. (2011). *T-Maze: a tangible programming tool for children*. Paper presented at the Proceedings of the 10th International Conference on Interaction Design and Children.
- Wiedenbeck, S., & Ramalingam, V. (1999). Novice comprehension of small programs written in the procedural and object-oriented styles. *International Journal of Human-Computer Studies*, 51(1), 71-87.
- Winslow, L. E. (1996). Programming Pedagogy—a psychological overview. *ACM SIGCSE Bulletin*, 28(3), 17-22.
- Wolber, D. (2011). *App inventor and real-world motivation*. Paper presented at the Proceedings of the 42nd ACM technical symposium on Computer science education.
- Wyeth, P., & Purchase, H. C. (2002). *Tangible programming elements for young children*. Paper presented at the CHI'02 Extended Abstracts on Human Factors in Computing Systems.
- Yau, J. Y.-K., & Joy, M. (2004). *Introducing Java: the case for fundamentals-first*. Paper presented at the International Conference on Education and Information Systems, Technologies and Applications (EISTA 2004).

Author Information

Can Mihci

Marmara University Institute of Educational Sciences,
Department of Computer Education and Instructional
Technology
Istanbul, Turkey
Contact e-mail: can.mihci@marmara.edu.tr

Nesrin Ozdener Donmez

Marmara University Ataturk Faculty of Education,
Department of Computer Education and Instructional
Technology
Istanbul, Turkey
