VISUAL VS. TEXTUAL PROGRAMMING: A CASE STUDY ON MOBILE APPLICATION PROGRAMMING BY TEENAGERS

[a]TOMÁŠ TÓTH, [b]GABRIELA LOVÁSZOVÁ

[a]Department of Informatics, Faculty of Economics and Management, Slovak University of Agriculture in Nitra, Tr. A. Hlinku 2, 949 76 Nitra, Slovakia
[b]Department of Informatics, Faculty of Natural Sciences, Constantine the Philosopher University in Nitra, Tr. A. Hlinku 1, 949 01 Nitra, Slovakia
email: [a]ttoth@uniag.sk, [b]glovaszova@ukf.sk

This study was written within the KEGA project 018UMB-4/2020 Implementation of New Trends in Computer Science to Teaching of Algorithmic Thinking and Programming in Informatics for Secondary Education.

Abstract: Choosing the right way of programming can prevent learning difficulties, contribute to increasing students' motivation to learn, and make teaching process more effective. The article is focused on assessing which way of programming, visual or textual, is appropriate for intermediate and advanced learners in the context of creating mobile applications. Three ways of programming were examined during an extracurricular programming course for teenagers aged 12-18 with previous programming experience and positive attitude to programming. The course was aimed at programming mobile applications. MIT App Inventor 2 as a visual programming tool and Android Studio with Java as a textual programming tool were chosen. Due to the gap between two programming tools, the method of transition from visual to textual programming using Java Bridge Code Generator and Java Bridge Library as mediators was implemented. The research results are based on the analysis of data obtained from participatory observations, interviews with students, questionnaires and source codes of applications created by students. The case study shows a difference in students' performance between visual and textual programming in favour of visual programming. However, the difference in students' attitudes toward visual and textual programming was the opposite in favour of textual programming, regardless of age and learning performance. These results suggest that App Inventor visual programming environment is advantageous at the beginning of learning programming, but may be perceived as too limited and not enough motivating for intermediate and advanced students, even though programming in Android Studio professional text-based environment is too challenging for them.

Keywords: mobile applications; teaching of programming; textual programming; visual programming.

1 Introduction

Teaching Informatics as school subject and specifically programming as a part of Informatics curriculum is important for students in several aspects. Computer skills are essential and beneficial for everyone in current digital age. However, teaching Informatics should not be focused only on acquiring skills to work with computers. Hromkovič and Steffen (2011) justify why teaching Informatics in schools is as important as other more traditional school subjects, thus it should also include fundamental concepts of computer science dealing with algorithmic information processing.

Teaching programming plays an important role in the development of computational thinking. This term was firstly introduced by Wing (2006): "Computational thinking involves solving problems, designing systems, and understanding human behaviour, by drawing on the concepts fundamental to computer science." It includes problem-solving skills such as abstraction, pattern recognition, decomposition, and algorithm design. Saeli et al. (2011) state that these skills are developed through programming, when students need to reflect how to communicate their solutions to the machine using a programming language. Several authors point to the benefits of learning programming to improve computational thinking and creativity through creating mobile apps (Dekhane et al., 2013; Tkáčová et al., 2017), designing games (Javidi and Sheybani, 2014), digital storytelling (Weintrop et al., 2018), controlling robots (Vega and Cañas, 2019). At the same time, difficulties with notation of algorithmic solutions need to be mentioned, which include problems of the syntax and the semantics of programming language that plays the role of a formal medium for expressing ideas.

Programming is considered by many authors as difficult and its learning is accompanied by various challenges. Many of them are associated with insufficient motivation of students, the ability to solve problems, or the choice of programming way and programming environment, such as:

- complexity of the programming language and programming environment (Koorsse et al., 2015; Krpan et al., 2017; Papadakis and Orfanakis, 2018; Radosevic et al., 2009; Saeli et al., 2011),
- difficulties with the basic programming concepts (e.g., control structures and loops) (Koorsse et al., 2015; Krpan et al., 2017; Mladenović et al., 2018; Ouahbi et al., 2015; Papadakis and Orfanakis, 2018; Radosevic et al., 2009),
- syntax and semantics issues (Koorsse et al., 2015; Krpan et al., 2017; Mladenović et al., 2018; do Nascimento et al., 2019; Ouahbi et al., 2015; Radosevic et al., 2009; Saeli et al., 2011),
- insufficient planning and designing of the algorithm (Koorsse et al., 2015; Krpan et al., 2017; Papadakis and Orfanakis, 2018).

Thus, the choice of an appropriate programming language and programming environment can affect students' success in their learning to program. Krpan et al. (2017) state that especially the student's first contact with programming is often a key moment when the student gains or loses interest in programming. For this reason, it is important to choose a suitable programming language and programming environment.

Garneli et al. (2015) also point out that many parameters must be considered in the teaching of programming, such as the age of the students, their experience, and the learning objectives. The right choice of programming way, programming language and programming environment can lead to the prevention of difficulties associated with learning programming and increase students' motivation to learn programming. Therefore, teaching environments developed especially for educational purposes are often used in education instead of professional programming environments and languages that are too complex for beginners.

Many educational programming environments use visual block-based program notation, which is considered more suitable for novice programmers than textual programming. João et al. (2019) present a cross-analysis of the core characteristics of 26 block-based and visual programming environments used in teaching computational thinking and programming. The overview presents wide range of visual programming environments suitable for age categories from preschoolers to high school students.

However, Deng et al. (2020) remind the fact that block-based programming is less authentic and less functional than text-based programming, and therefore, block-based programming alone might not be enough for students to understand the real meaning of programming and may have a negative impact on their future studies in computer science. Noone et al. (2021), at the same time, point out that there exists a gap in the education of students in their mid-to-late teenage years, when perhaps visual programming languages are no longer suitable, but textual programming languages may involve excessive learning effort.

While visual programming is considered to be more advantageous choice for novice programmers (Attard and Busuttil, 2020; Deng et al., 2020; Weintrop and Wilensky, 2017), various studies address the process of transition from visual to textual programming for intermediate and advanced students (Cheung et al., 2009; Krpan et al., 2017; Noone et al., 2021; Vega and Cañas, 2019; Weintrop and Wilensky, 2019).

This article explores one implementation of a transition from visual to textual programming in the context of mobile application development, which uses hybrid environment for bridging the gap between visual and textual programming, and answers the following research questions (RQs):

RQ1: How does performance of intermediate teenage programmers in the field of creating mobile applications differ according to the way of programming (visual, hybrid, textual)?

RQ2: How do teenagers' attitudes toward learning programming in visual and textual way differ according to age and programming skills?

In our implementation, MIT App Inventor visual programming environment, Java Bridge Code Generator hybrid environment, and Android Studio with Java Bridge Library textual programming environment are used during informal programming course for teenage students with previous programming experience.

2 Ways of Code Creating

2.1 Visual Programming

In visual programming, the programming is performed using a visual programming language and a visual programming environment. The visual programming language is made up of pre-prepared graphic elements. Each graphic element represents a certain part of the programming language – individual commands, programming concepts (e.g., command to create a variable, loop, condition). The graphic elements also use a higher degree of abstraction, thanks to which even more complex functions can be encapsulated in one graphic block. Therefore, the programmer does not need to know how the function is implemented in order to be able to work with it (Paternò and Santoro, 2019).

Programming is done by combining pre-prepared graphic elements. Graphic elements (also called blocks) are usually connected in a drag & drop way – the programmer takes a specific block from the palette (block menu) and moves it to the canvas (desktop). So, it is not necessary for the programmer to memorize commands (Krpan et al., 2017; do Nascimento et al., 2019; Weintrop, 2015). When connecting graphic blocks, it is defined which blocks can be and which cannot be connected to each other. This is usually ensured in programming environments by using the principle of jigsaw puzzle, where the graphic blocks have a shape like parts of a puzzle (Hsu, Ching, 2013; Musmarra, 2018; Paternò and Santoro, 2019). It makes joining blocks more intuitive (Weintrop and Wilensky, 2019).

If two blocks cannot be joined into the syntactically correct form of the expression, the programming environment prevents them from being joined. It is a prevention of syntactic errors (Hsu, Ching, 2013; Koorsse et al., 2015; Weintrop and Wilensky, 2018). The shape of graphic block is also a hint of how many connections with other blocks can be made in terms of inputs as well as outputs of the given block (Paternò and Santoro, 2019). Weintrop (2015) states that although visual programming environments prevent the creation of syntax errors, overall, they do not solve this problem, but only delay it to later periods of programming in other textual programming languages.

Visual programming and visual programming environments are currently popular especially in teaching the programming fundamentals. Their use is successful in involving students in programming activities and providing a sense of success in the early stages of learning to program. Such languages are mainly used for the development of algorithmic thinking. Examples of visual programming languages and environments are Scratch, MIT App Inventor 2 and Alice. An overview of pros and cons of visual programming is presented in Table 1.

2.2 Textual Programming

In textual programming, the programming is performed by writing text and with using textual programming environments. A textual programming language is a programming language which consists of a set of instructions that are in the textual form. All textual programming languages have their own syntax rules. In the case of this type of programming languages, the creation of semantically and syntactically correct code is not ensured by such mechanisms as in the case of visual programming languages. This fact increases the complexity of programming. Unfortunately, some errors in the text code may be reflected in the incorrect functionality of the program, or the program may not even be compiled and run, unlike visual programming, where the wrong code essentially cannot be created. Such situations require extra effort and time to properly identify the error in the program and resolve it. Therefore, textual programming languages can be challenging for the students. For novice programmers the complexity of textual programming and the number of commands is often limiting in creating algorithms. However, the disadvantage of more demanding program creation can be overcome by students' feeling that they are working with a professional tool. Students can gain an authentic programming experience. Such programming opportunity may be interesting for students with greater expectations and needs (Garneli et al., 2015; Mladenović et al., 2018). Examples of textual programming languages are Python, Java, and C#. An overview of pros and cons of textual programming is presented in Table 2.

Table 1 Pros and cons of visual programming

Pros	Cons
▪ Easy to start creating functional programs ▪ A large amount of knowledge is not required (general about programming, to memorize commands) ▪ Intuitiveness of program creating by joining blocks (prevention of syntax errors as well) ▪ The graphic nature increases the intelligibility of the elements ▪ Immediate feedback ▪ Simplified error detection ▪ Attractiveness for students (it consists in the interactivity of programming environments, in the use of multimedia elements and in the thematic focus)	▪ Deteriorating readability / comprehensibility of the program with increasing program complexity ▪ Limited options of program creating (some features or options may not be available)

Table 2 Pros and cons of textual programming

Pros	Cons
▪ More suitable for creating more complex programs ▪ The readability of the program can be maintained even with its increasing complexity ▪ More options for creating a program (the programmer is not as limited as in the case of a visual programming language)	▪ More difficult to understand for beginners ▪ Requires more knowledge (general about programming, know commands, syntax)

2.3 Hybrid Programming

The meaning of the term hybrid programming in the context of this article cannot be confused with the meaning of this term in context of cross-platform mobile app development (the application is developed for several operating systems at the same time) nor with the term in context of multi-paradigm programming languages (programming languages based on more than one programming paradigm). In the connection with visual and textual programming, hybrid programming is a such way of programming where the students use both elements of visual programming and textual programming. The student can create program by joining graphic blocks, while the visual code can be translated into an equivalent textual form and the textual code can be modified and expanded by writing text – program instructions in textual form.

It is usually used in the student's transition from visual to textual programming as an intermediate step between these two ways of programming. Students can better create mental connections between these two contexts (visual and textual programming) by using both ways of programming at the same time and working with visual and textual representation of the same program. The aim is to make this transition easier for students, to make it smoother and to prevent various difficulties, which are associated with the transition (Tóth and Lovászová, 2018). An overview of pros and cons of hybrid programming is presented in Table 3.

Table 3 Pros and cons of hybrid programming

Pros	Cons
▪ Possibility to work with visual and textual programming ▪ Helping to create mental connections between visual and textual programming ▪ Simplifying the student's transition from visual to textual programming	▪ Requires knowledge of visual as well as textual programming ▪ Greater complexity of program creation due to the use of two ways of programming

Examples of hybrid programming languages, environments, or tools which support hybrid programming are:

- Java Bridge – uses visual programming of mobile applications on the principle of MIT App Inventor 2 and textual programming in the Java programming language (App Inventor (a), n.d.),
- PencilCode – allows to create and edit code in a textual way and at the same time with graphic blocks (Alrubaye, 2019)
- Pencil.cc – an environment that allows the creation of isomorphic code by visual and textual programming (Weintrop and Wilensky, 2017),
- PyBlockly – the environment based on the principle of turtle graphic; it uses visual programming language and textual programming language Python (Strong et al., 2018),
- BrickLayer – allows to visually create programs for Arduino microcontrollers platform and the code is translated into the textual programming language C (Cheung, 2009).

3 Materials and Methods

3.1 Implementation Process

Based on the defined research questions, we carried out research in the field of teaching mobile application programming in secondary education. For this purpose, a leisure course of mobile application programming was organised. Applications were created for operating system Android. The research was conducted during the school year 2018/2019. The course took place once a week and comprised two school lessons (total of 90 minutes). The course was led by one lecturer, who was also in the role of researcher. All conceptual and teaching issues were consulted with expert researcher.

The intention of the course was also to allow students to program in all three ways of programming. The aim was to verify the suitability of the implementation of these programming ways in secondary education and their impact on the effectiveness of education. The course schedule was divided into three stages:

1. Visual programming stage

- students use visual programming in visual programming environment,
- education is focused on basic programming concepts,
- getting to know the programming environment and gaining the first experience in creating mobile applications by visual programming.

2. Hybrid programming stage

- first, students use visual programming in visual programming environment,
- subsequently, students generate equivalent text code from the code in the form of graphic blocks,
- the generated text code is transferred to the textual programming environment,
- education is focused on getting to know the text equivalent of an already known program previously created by visual programming, getting to know the new used tools, textual programming environment and textual programming language,
- students experiment with minor modifications of the code by textual programming (e.g., changing the arguments of commands), analogically extend the code by textual programming according to the already generated textual code,
- gaining the first experience with development in a textual programming environment.

3. Textual programming stage

- students use textual programming in textual programming environment,
- pointing to the analogy with programming in visual programming environment; techniques of mediated knowledge transfer from one context to another one are used too (Perkins and Salomon, 1988; Perkins and Salomon, 1992).

The order of the stages was deliberately chosen. Visual programming is easier to get started, so it was included in the first stage. The direct transition from visual to textual programming can be accompanied by various challenges, so the stage of hybrid programming has been inserted between visual and textual programming stage as an intermediate step. The design of the transition strategy from one way of programming to another one is dealt in more detail in (Tóth and Michaličková, 2018). In each stage, students worked on three projects (Table 4). The difficulty of projects increased during the stage.

Table 4 The course schedule

Stage	1. Visual programming			2. Hybrid programming			3. Textual programming		
Project	Hello World (V1)	Catch the Egg (V2)	Project (V3)	Hello World (H1)	ChatBot (H2)	MoleMash (H3)	Hello, Purr (T1)	Roll the Dice (T2)	Project (T3)
Duration (number of lessons)	1	3	5	3	3	3	1	2	5
Complexity	Simple	Simple/ Complex	Complex	Simple	Simple/ Complex	Complex	Simple	Simple	Complex

In addition to the personal assistance of the lecturer, the students had available short handouts as another assistance tool. The handouts contain application instructions, such as application functionality requirements, a preview of the application graphical user interface (GUI), an outline of the solution in form of subtasks and others. The purpose of using short handouts as learning material was also to support students in active independent work on projects. At the same time, it allowed differentiation according to students' abilities.

Students could demonstrate independence at work especially by solving individual projects. The creation of individual projects was included at the end of the stage of visual and textual programming. The topic of the individual project was chosen by the students in both stages. By creating the individual project, students had to prove what they learned in the previous period and what application they are able to create independently. Unlike the previous created applications, during the lessons on which the students worked on their own individual projects, they did not have a formal description of the final product and sketch of the solution. The assignments of individual projects were formulated in such way to provide maximum space for students for their own creativity and creation. During the application creation they had to analyse the problem, design the structure of the application and design how to implement its functionalities.

3.2 Participants

The research is carried out with a small group of participants, in which examined elements are recorded in detail and analysed. The research sample consists of 14 secondary school students. 13 students were male and one female. Students' age range were from 12 to 18 years.

In order to determine students' attitude to programming, the range of programming experience, and knowledge of basic programming concepts, an entry questionnaire was prepared. The students' answers show that:

- Students have a positive attitude towards Informatics and programming. They attended our programming course in their leisure time. Their increased interest in programming is also evidenced by the fact that 86% of students stated that they enjoy programming very much and the remaining 14% stated that they enjoy programming a bit. 57% of students stated that they would like to devote to programming at a professional level in the future.
- All students already had programming experience. Half of the students program one or two years. Students already had experience mostly with educational programming environments and languages such as Imagine Logo, Scratch, Python and Baltík. Four students also had experience with MIT App Inventor 2, two students had experience with the Java programming language and one student had experience with Android Studio. No student in the research sample had experience with Java Bridge. Some students also had experience with programming tangible construction kits and robotics (e.g., Lego Mindstorms, Micro:bit, Sphero, Ozobot) and four students also had some experience with programming mobile devices.
- Students are familiar with several terms in the field of algorithmic structures (loop, conditions, procedure, library), work with data (variable, parameter, constant, data type), object-oriented and event-driven programming (class, object, event, component). The level of conceptual understanding was not ascertained.

3.3 Instruments

The focus on programming mobile applications also influenced the choice of programming environment for the stages. For each of the three stages, we chose environment which allows mobile application programming and at the same time it allows programming in the way specified for the stage. The selected tools can be divided according to the stage in which they were used:

1st stage: MIT App Inventor 2 (MIT AI 2)

- visual programming environment for creating applications for mobile devices with operating system Android,
- hides the complexity of development and allows the student to focus on the design GUI of application, its functions and how the user will work with it.

2nd stage: Java Bridge (Java Bridge Code Generator) and Android Studio

- under the term Java Bridge is distinguished Java Bridge Code Generator and Java Bridge Library,
- Java Bridge Code Generator is an exploratory version of the programming environment MIT AI 2 that allows students to create an application just like in programming environment MIT AI 2 (by visual programming) and then generate an equivalent textual version of the application code in the programming language Java (App Inventor (b), n.d.),
- Android Studio is used to view and edit the generated code.

3rd stage: Android Studio, Java and Java Bridge (Java Bridge Library)

- Android Studio is a professional textual programming environment for creating mobile applications for the operating system Android,
- Java is a programming language for programming mobile applications for the operating system Android,
- Java Bridge Library is a library of programming language Java,
- Java Bridge Library uses the same terminology as is used in MIT AI 2 – there is a Java class for each component – the class encapsulates the complexity of functionality just like in MIT AI 2,
- Java Bridge Library in this way facilitates textual programming of mobile applications for operating system Android than it is with standard way using Android SDK (App Inventor (b), n.d.).

3.4 Data Collection and Data Processing

Several research methods were used for data collection: questionnaires, problem-solving interviews, informal interviews with students, focus groups, participatory observation, unstructured observation, field notes and product collection (student-created applications).

Using these data collection methods, we obtained data which were processed by qualitative and quantitative methods. Therefore, some qualitative data were quantified (converted to numerical form). We obtained data of three types:

1. data obtained from observations and interviews

- converted into text in the form of protocols,
- texts were analysed and processed by categorization and coding,

2. data obtained in textual form from questionnaires

- entry questionnaire implemented using online Google Forms about students' attitudes, motivations, and their aptitude for programming
- questionnaires during lessons implemented using Socrative audience response software with instant feedback after each question used to:
  - verify student's knowledge and understanding of the crucial concepts,
  - get continuous feedback on students' attitude to the content and the form of lessons
- final questionnaire implemented using online Google Forms containing questions on self-assessment and on attitudes to the programming tools and approaches used.

3. data obtained in the form of collected products – applications' source codes created by students

- to verify the student's mastery of the problem and to identify problematic parts of its solution,
- the source codes were uploaded to the cloud storage Google Disk by the students for making them available to the lecturer,
- the source codes were analysed, and the obtained data were quantified into:
  - *difficulty score* – the sum of *programming difficulty* (the number of essential activities related to designing and coding) and *technical difficulty* (the number of essential activities related to the project development in the programming environment and to building the application),
  - *solution success rate* – the extent of learning objectives defined for the project achieved by student,
  - *weighted performance* – students' performance in solving projects; evaluated on the basis of difficulty score and solution success rate.

Collected data was coded by two researchers and analysed through discussion.

4 Results

The achieved results are divided into results obtained from subjective data (from observations and expressions of students) and into results obtained from objective data (from the analysis of submitted products).

4.1 Results from the Evaluation of Subjective Data

Results from subjective data are evaluated according to the defined stages.

4.1.1 Stage of Visual Programming

The visual programming environment MIT AI 2 did not cause problems for the students. The students advanced quickly. They were able to solve tasks independently. Thanks to the handouts, students were able to work at their own pace. They were able to perform at the level of Creativity of the Revised Bloom's Taxonomy already during the creation of the applications which were created by students together with the lecturer. The students' activity increased even more in solving individual projects. They worked creatively and, in addition, improved the applications with various personal ideas. Errors in the program occurred only occasionally, mainly related to the application logic.

The mobile applications development in MIT AI 2 was enjoyed by students. The students presented their experience with MIT AI 2 as positive, what is also confirmed by the students' verbal statements:

Student12: *„I like that it was quite easy to program there (in MIT AI 2).“*

Student14: *„I liked the ease with which applications could be created.“*

Student4: *„I liked it, I'm glad I learned how to program applications in App Inventor.“*

Students' interest in programming using MIT AI 2 is also confirmed by other students' statements. A total of 85.7% of students stated in the final questionnaire that they programmed applications in MIT AI 2 on their own initiative at home as well.

Most students feel confident developing mobile applications in MIT AI 2. A total of 75% of students answered that they can create applications in MIT AI 2 and no student stated that they cannot create applications (Table 5). The connections of the answers with the students' age were not recorded in this case.

While working with MIT AI 2, the students gradually encountered several limitations in programming the application functionality or creating GUI, such as the inability to dynamically create GUI components, or limited options for setting up components. In addition, students began to express feelings and opinions that MIT AI 2 is already easy for them:

Student9: *„App Inventor is so childish.“*

The students were interested in moving even further forward in programming, including by moving to another programming environment:

Student1: *„It was a nice introduction to mobile application programming, but it's time to move on.“*

4.1.2 Stage of Hybrid Programming

Thanks to the handout, students were able to work independently at this stage too. Some students solved all the tasks and programmed the application according to the instructions from the handout even without the lecturer help. A few students even expanded the application with more similar features.

The students also did well with programming in Android Studio, even though they did not know the meaning of each line of Java code. They were able to solve tasks logically and analogously according to the already existing code and their previous experience and knowledge. We also noticed a positive attitude from the student's statement:

„It looks complicated, but it's quite easy to understand.“

The youngest students aged 12 and 13 progressed the slowest. The greatest progress was made by Student7 (16 years old). This student was so successful in textual programming that he did not even use the Java Bridge Code Generator and he did not program in hybrid way. He programmed using just textual programming and Java Bridge Library. Later, several other students gradually joined this student. Student7 explained his action by saying that creating an application using textual programming does not cause him a problem. On the contrary, the combined work with visual and textual programming environment is delaying.

Syntactic errors did not occur much at this stage. Only the youngest student had the biggest problems with syntactic rules. The most common problems were technical:

- Problems related to project as an application structure – students had difficulty understanding the nature of using the project as a whole covering different parts of the application (e.g., problem to distinguish where to insert images within

the folder structure of the project, problem to distinguish between the meaning of the project and the Java file).

Table 5 Students' subjective evaluation of the degree of mastery of mobile applications creating in MIT AI 2

Degree of mastering the creation of mobile apps in MIT AI 2	Likert scale (LS)	Number of students by age							Total	Average age
		12	13	14	15	16	17	18		
I handle it very well	2	0	0	1	1	0	0	0	2 (25%)	14.5
I can handle it	1	1	0	1	0	1	1	0	4 (50%)	14.8
Undecided	0	0	1	0	0	0	0	1	2 (25%)	15.5
I cannot handle it	-1	0	0	0	0	0	0	0	0 (0%)	N/A
I cannot handle it at all	-2	0	0	0	0	0	0	0	0 (0%)	N/A
Average answer (LS)					1.0					-

- Problems related to Java code structure – to which specific place in the Java code student should write the new code (it is necessary to be aware of the code structure, ranges of code parts (enclosed in curly brackets {}), but also which parts of the code refer to each other (which methods are called and from which part of code are called)). These problems occurred mainly in the case of the three youngest students.

The students began to feel frustration and demotivation due to higher incidence of errors and the need to resolve them. Debugging errors of various kinds required much more time and more attention of students at this stage. The development of the application was no longer so smooth and linear. The help of a lecturer was more necessary.

Despite these difficulties students expressed a positive attitude towards the Java Bridge Code Generator in their final assessment (Table 6). The connection of the answers with the students' age was not recorded in this case.

The students positively commented Java Bridge Code Generator, but they also were able to critically evaluate it:

„I liked that the Java code could be generated."

„I could help myself if I couldn't program something in Java."

„Easy transfer of the code to other programming environments, but I lack the ability to run it quickly (build, run and test application)."

„I didn't like that the code generation feature wasn't working as it should and you still need to modify the application code to work."

When asked whether the code generation was helpful for students together more than half of the students (57.2%) gave a positive answer (Table 7).

4.1.3 Stage of Textual Programming

Already during the transition to textual programming of mobile applications, students had positive expectations. Students again became more successful in application development. In contrast to the end of the second stage, where there were feelings of frustration and demotivation, in the third stage, the students began to make a positive impression again. All students managed the first application without major problems. Essentially, the students were able to work independently according to the handout. However, there were bigger differences in their pace of work. Some students also worked at home in their own initiative according to handout (stated by 50% of students in questionnaire).

If necessary, students also helped themselves using Java Bridge Code Generator and hybrid programming at this stage, especially when solving individual projects (T3) – stated by total of 42.9% of students in final questionnaire (Table 8).

They used Java Bridge Code Generator mainly in case of:

- GUI creation – for more convenient GUI creation visually using virtual screen than textually using the Java Bridge Library in Android Studio,
- if they forgot how to write the command – e.g., what the command syntax for defining the event listener looks like.

Table 6 Students' evaluation of their attitude to Java Bridge Code Generator

Students' attitude to Java Bridge Code Generator	Likert scale (LS)	Number of students by age							Total	Average age
		12	13	14	15	16	17	18		
Great tool	2	0	2	0	2	0	1	1	6 (43%)	15.2
Good	1	0	0	0	1	1	1	0	3 (21%)	16.0
Undecided	0	1	0	0	0	0	0	1	2 (15%)	15.0
Poor	-1	0	0	2	0	1	0	0	3 (21%)	14.7
Very bad tool	-2	0	0	0	0	0	0	0	0 (0%)	N/A
Average answer (LS)					0.9					-

Table 7 Students' evaluation of rate of help provided by code generation

Rate of help provided by code generation	Likert scale (LS)	Number of students by age							Total	Average age
		12	13	14	15	16	17	18		
Definitely helpful	2	0	1	0	1	1	2	1	6 (42.9%)	16.0
Rather helpful	1	0	1	1	0	0	0	0	2 (14.3%)	13.5
Neither yes nor no	0	0	0	0	2	0	0	0	2 (14.3%)	15.0
Rather not helpful	-1	1	0	0	0	1	0	1	3 (21.4%)	15.3
Not helpful at all (it is not needed)	-2	0	0	1	0	0	0	0	1 (7.1%)	14.0
Average answer (LS)					0.6					-

Table 8 Students' evaluation of the frequency of helping with the code generation during textual programming of individual project (T3)

The frequency of helping with the code generation during programming of individual project (T3)	Likert scale (LS)	Number of students by age							Total	Average age
		12	13	14	15	16	17	18		
Often	2	0	0	0	0	1	1	0	2 (14.3%)	16.5
Occasionally	1	0	0	1	1	0	1	1	4 (28.6%)	16.0
Undecided	0	0	0	0	0	0	0	0	0 (0%)	N/A
Rarely	-1	1	1	0	0	0	0	0	2 (14.3%)	12.5
Never	-2	0	1	1	1	0	0	0	3 (21.4%)	14.0
I did not program T3	-	0	0	0	1	1	0	1	3 (21.4%)	16.3
Average answer (LS)					0					-

A greater connection with the students' age was recorded in this case – especially younger students said they did not help themselves by generating code – some were discouraged by the complexity of using two programming environments, for others such work was delaying.

The students faced several challenges when programming mobile applications in textual way compared to visual programming:

- Work also at the level of the file and folder structure of the project – it was no longer enough to work only at the level of creating the code (e.g., creating a screen consists not only of creating the screen as a logical element in the code, but also as a file).
- Work with multiple files at once – when implementing some functions of the application, it is necessary to intervene in the code of several project files (e.g., add vibration command in .java file and add vibration permission in AndroidManifest.xml).
- Work with files in multiple formats – students worked with .java, .xml files and with media of various formats during the development.
- GUI creation in textual way – GUI is not created with Java Bridge Library in visual way by moving components to the virtual screen of the device using drag & drop method; GUI is created by writing textual code within a .java file.
- Syntax of the programming language – students must know and consciously follow the syntactic rules of the programming language.
- Writing code directly – there is no such pallet of components and blocks as in MIT AI 2, from which students would just choose and compose a program.
- Creating responses to the events – multi-step implementation requiring code to be written to multiple locations within the code structure of the .java file.
- Working with data types – when creating variables and objects, it is necessary to define their data type.

The errors that most often occurred to students during programming at this stage can be categorized as follows:

- problem with application building due to the use of different physical devices for testing by students (need for proper configuration of the build),
- omission of some part of the implementation of the event response or incorrect definition,
- adding a command to the wrong place within the structure of the .java file, or due to the semantic meaning of the commands (e.g., adding a command to open a new window to the part of another method where the method parameters should be written) or with respect to the chronological execution of the code (e.g., first the action was performed according to the generated number and only then the number was generated, or the use of a component that was only declared and not initialized),
- syntactical errors (e.g., missing semicolon, brackets, etc.) – already higher incidence than in the previous stage.

Despite several complications and a higher incidence of errors during the third stage of textual programming, in the end, the students were not dominated by negative feelings or attitudes to textual programming in Android Studio. Their feeling and attitudes were exactly the opposite. For example, although working with a project at the level of its folder structure appeared to be problematic for students during the application creation, only one student confirmed this in final questionnaire. A total of 64.3% of all students said that working with the project is not difficult. Most students do not even find textual programming in Android Studio difficult at all (Table 9).

Similarly, the students expressed a positive self-assessment of their ability to create applications by textual programming in Android Studio using Java Bridge Library. 35.7% of students stated that they can create an application in Android Studio completely independently. The students most often stated that they can create the application with the help of more experienced person (e.g., a teacher) (57.1% of all students stated this), with the help of internet (50%), with the help of the handout (42.9%) and with the help of code generation (42.9%). None of the students stated that he/she is not able to create application in Android Studio.

Similarly, the students expressed a positive self-assessment of their ability to create applications by textual programming in Android Studio using Java Bridge Library. 35.7% of students stated that they can create an application in Android Studio completely independently. The students most often stated that they can create the application with the help of more experienced person (e.g., a teacher) (57.1% of all students stated this), with the help of internet (50%), with the help of the handout (42.9%) and with the help of code generation (42.9%). None of the students stated that he/she is not able to create application in Android Studio.

In addition, half of the students stated that textual programming suited them best and on the contrary, no student stated that the visual programming suited him/her best (Table 10). The distribution of the answers is not related to age.

Due to this subjective perception of students, students stated that they would welcome even more opportunities to deal with textual programming in programming language Java and even without using the Java Bridge Library.

Table 9 Students' difficulty evaluation of textual programming in Android Studio

Difficulty of textual programming in Android Studio	Likert scale (LS)	Number of students by age							Total	Average age
		12	13	14	15	16	17	18		
Very easy	2	0	0	1	0	1	0	0	2 (17%)	15.0
Rather easy	1	0	0	1	1	0	0	0	2 (17%)	14.5

Neither easy nor hard	0	0	2	0	1	1	2	1	7 (58%)	15.6
Rather hard	-1	1	0	0	0	0	0	0	1 (8%)	12.0
Very hard	-2	0	0	0	0	0	0	0	0 (0%)	N/A
Average answer (LS)					0.4					-

Table 10 Students' evaluation of their affection for a specific way of programming

Students' affection for a specific way of programming	Number of students by age							Total	Average age
	12	13	14	15	16	17	18		
Textual	1	1	2	1	1	1	0	7 (50%)	14.4
Hybrid	0	1	0	1	0	1	1	4 (28.6%)	15.8
Visual	0	0	0	0	0	0	0	0 (0%)	N/A
Undecided	0	0	0	1	1	0	1	3 (21.4%)	16.3

4.2 Results from the Evaluation of Objective Data

Based on a detailed analysis of data in the form of submitted products (programming projects created by students), we have achieved the results of objective evaluation of students. In terms of the difficulty score, the following facts can be observed (Table 11):

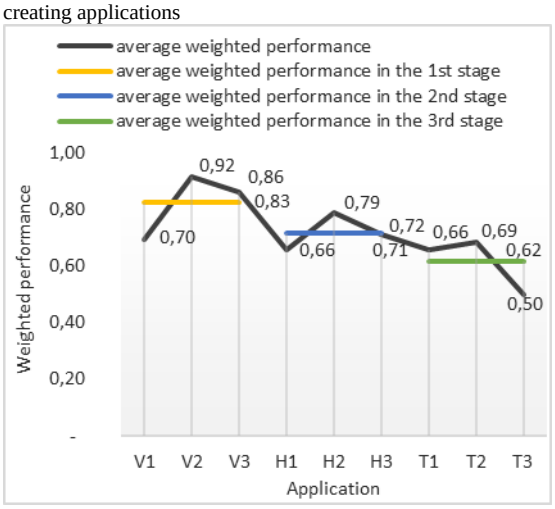
- the projects in the 2nd stage had the highest average score of the total difficulty and to lowest in the 3rd stage,
- the projects in the 2nd stage had the highest technical difficulty score and the lowest in the 1st stage,
- the projects in the 1st stage had the highest programming difficulty score and the lowest in the 3rd stage.

In terms of the average weighted performance of students (Figure 1), it can be stated:

- students on average reached the highest value of the average weighted performance in the 1st stage and the value decreased with each subsequent stage (0.83 0.72 → 0.62),
- at the beginning of each stage, it is possible to observe an increase in weighted performance; the most significant increase was at the beginning of the 1st stage; the increase at the beginning of the stage decreases with each subsequent stage,
- the average weighted performance in the individual project in the 1st stage (V3) is greater than in the case of the individual project in the 3rd stage (T3).

Taking a more detailed look at the distribution of students' weighted performance in creating individual projects in the first (Table 12) and the third stage (Table 13) according to age, we can observe that there are no significant age differences in student performance when programming in App Inventor. In contrast, when programming in Android Studio, the weakest performance is reached by the three youngest students.

Graph 1 Average weighted performance of students during creating applications



5 Discussion

Based on the obtained results, we formulate answers to the research questions.

RQ1: How does performance of intermediate teenage programmers in the field of creating mobile applications differ according to the way of programming (visual, hybrid, textual)?

Results showed that students achieved the highest average weighted performance in the visual programming stage. At this stage, students also made the most significant progress (the most significant increase of average weighted performance) of all three stages (Figure 1). These factors have a positive effect on students' sense of success in mobile application programming.

Table 11 Difficulty score of projects in the visual, hybrid, and textual stage

Project identifier	1st stage (visual)			2nd stage (hybrid)			3rd stage (textual)		
	V1	V2	V3	H1	H2	H3	T1	T2	T3
Programming difficulty	8	17	17.2	9	10	15	7	10	10.9
Technical difficulty	7	5	4.8	10	11	13	8	8	7.1
Total difficulty	15	22	22	19	21	28	15	18	18
Average difficulty		19.67			22.67			17.00	

Table 12 Weighted performance of students in individual projects V3

Weighted performance in V3	Number of students by age							Total	Average age
	12	13	14	15	16	17	18		
1.50 – 1.21	0	0	1	0	0	0	0	1 (7%)	14.0
1.20 – 0.91	1	0	0	0	0	1	0	2 (14%)	14.5
0.90 – 0.61	0	0	1	2	1	1	1	6 (43%)	15.8
0.60 – 0.31	0	2	0	0	0	0	1	3 (22%)	14.7
0.30 – 0.00	0	0	0	0	0	0	0	0 (0%)	N/A
no rating (not submitted)	0	0	0	1	1	0	0	2 (14%)	15.5

Table 13 Weighted performance of students in individual projects T3

Weighted performance in T3	Number of students by age							Total	Average age
	12	13	14	15	16	17	18		
1.50 – 1.21	0	0	0	1	0	0	0	1 (7%)	15.0
1.20 – 0.91	0	0	1	0	0	0	0	1 (7%)	14.0
0.90 – 0.61	0	0	0	0	1	0	1	2 (14%)	17.0
0.60 – 0.31	0	0	0	0	0	1	0	1 (7%)	17.0
0.30 – 0.00	1	1	1	0	0	0	0	3 (22%)	13.0
no rating (not submitted)	0	1	0	2	1	1	1	6 (43%)	15.6

Students had a high level of self-confidence in visual programming – no student stated that he/she cannot create applications in MIT AI 2. The created applications in this stage had the lowest technical difficulty score and, conversely, the highest programming difficulty score (Table 11), which indicates a low workload of students with technical aspects of the development and high degree of programming skills to work in this way. Students were able to focus mainly on programming itself and not on the technical aspect of development. These results are consistent with the statements of other researchers that are presented in the chapter Ways of Code Creating. Visual programming enabled students to make great use of their own creativity in creation. Looking at the results of creating individual projects, which reflect the level of students' ability to independently create an application in a specific way of programming, we can see that students also achieved the highest degree of independence in visual programming – the average weighted performance in projects V3 is higher than in T3 (Figure 1).

During hybrid programming, there were greater differences between students in their ability to move forward and create applications in this way. Students' performance was also negatively affected by the fact that the applications created at this stage had the highest difficulty score. Especially the difficulty of technical aspect of the solution increased (the highest technical difficulty score) – also caused by using two programming environments simultaneously. Despite the greater incidence of difficulties in hybrid programming than in visual, the most students subjectively rated the Java Bridge Code Generator positively (Table 6). Students marked code generation as helpful in creating applications and no correlation between responses and students' age was recorded (Table 7).

Textual programming was a challenge for students. Higher complexity of creating applications was reflected in:

- the lowest achieved difficulty score of created applications (Table 11),
- the lowest achieved average weighted performance (Figure 1),
- more significant differences between students in weighted performance during their independent work in T3 than V3 (Table 12 and Table 13),
- lower achieved weighted performance in the case of projects T3 than V3 for each student (with one exception).

While visual programming was mastered by all students, regardless of their age, in textual programming it is not possible to say so clearly. The weakest results were achieved especially by the youngest students with least experience.

Our findings that students performed better in visual than in textual programming are consistent with previous studies in sense that students who use block-based programming tools outperform the students who use textual programming tools (Deng et al., 2020; Weintrop and Wilensky, 2017].

RQ2: How do teenagers' attitudes toward learning programming in visual and textual way differ according to age and programming skills?

Despite the results that students' learning outcomes in visual programming was better than that in textual programming, the subjective perception of benefits of visual programming by students was not so definite regardless of students' age and programming skills:

- although working with the project at the level of the folder structure appeared to be problematic, the students did not confirm such a perception with their own statements – only one student stated it as difficult,
- students do not find textual programming in Android Studio difficult – only the youngest student commented that it is difficult (Table 9),
- none of the students said that he/she cannot create the application in Android Studio,
- 50% of students said that they were most comfortable with the textual way of programming in Android Studio (no student mentioned visual programming in MIT AI 2) (Table 10).

These findings are in compliance with Weintrop and Wilensky (2017) who found no difference between students learning in block-based and text-based conditions with respect to confidence or enjoyment. Comparably to our results, authors report that students who program in textual way considered their programming experience as more similar to what professional programmers do and as more effective at improving their programming abilities. Our results are also in line with teachers' experience and views investigated by Attard and Busuttil (2020) that using an interface such as App Inventor would attract students immediately due to its visual nature as opposed to text-based languages such as Java, but could be too limited for intermediate and advanced learners.

The motivation to program in a hybrid way was mainly to help with the textual programming. On the other hand, especially younger students were discouraged from such assistance by more complicated combined work with two programming environments simultaneously (Table 8). The motivation of students to program in a hybrid way also decreased with the acquired experience of students – the work with the two environments simultaneously was delaying for students. A different result could be recorded in the case of using a mediation tool, in which the possibility of programming in a visual and textual way is integrated within one programming environment.

Textual programming of mobile applications in professional programming environment Android Studio proved as a great challenge for students. Already during the first stage of visual programming, students expressed interest in the transition to such more professional way of programming. Despite the weaker measured objective results of students in textual programming, the positive subjective perception of their work persisted. This can be attributed to the high degree of motivation to create applications in this way regardless of age. This result coincides with the result achieved in our antecedent exploratory research (Tóth and Lovászová, 2018). In addition, students were motivated to continue textual programming of mobile applications even without the Java Bridge Library as assistance tool.

The following limitations should be considered when interpreting the results:

- Selection of research sample – a smaller number of participants allowed us to focus on a deeper understanding of the observed phenomenon. On the other hand, it is not possible to generalize the results to all students of secondary education. The research sample consisted of a selection of students with an increased interest in computer science and programming. In the case of a common sample of students, the results could deviate from ours.
- Choice of textual programming language and programming environment – to create mobile applications for the operating system Android, the Java programming language and the professional programming environment Android Studio were chosen. Choosing a programming language and programming environment more appropriate for teaching introductory programming could also affect results.

6 Conclusion

The aim of the article is to assess which way of programming, visual or textual, is appropriate for intermediate and advanced learners in the context of creating mobile applications. A case of teaching programming within the extracurricular course intended for students interested in creating mobile applications has been presented and studied. Based on the qualitative analysis of source codes, students' performance in visual, hybrid, and textual way has been evaluated. Furthermore, students' attitudes to the used ways of programming have been examined too.

The results showed that students were able to achieve better performance using visual programming than the other two ways of programming regardless of their age. In the case of textual programming, students' performance differed according to age. The weakest performance was achieved especially by the youngest students with the least experience. Regarding attitudes toward the way of programming, all students declared positive perception of textual programming in Android Studio despite many challenges they had to overcome. Hybrid visual/textual programming was used in order to help the transition between visual and textual programming. However, besides positive aspects of using hybrid tool in helping to generate textual code, combined use of two programming environments simultaneously during hybrid programming was perceived by students as complicated and delaying.

In the future, replication of this research under modified conditions may contribute to the problem of determining the appropriate way of learning programming mobile applications. The proposed modification of conditions is using a more comfortable programming tool with integrated visual and textual programming at the same time, which prevent difficulties with the complicated use of several tools at the same time. Furthermore, replication with a sample of students from common class would yield valuable results for the area of formal secondary education.

Literature:

1. Alrubaye, H., Ludi, S., Mkaouer, M.W.: Comparison of Block-Based and Hybrid-Based Programming Environments in Transferring Programming Skills to Text-Based Environment. In CASCON '19: Proceedings of the 29th Annual International Conference on Computer Science and Software Engineering, 100-109.
2. App Inventor (a): App Inventor Java Bridge. Available online: http://www.appinventor.org/jbridge.
3. App Inventor (b): Java Bridge Programming. Available online: http://www.appinventor.org/jBridgeIntro.
4. Attard, L., Busuttil, L.: Teacher Perspectives on Introducing Programming Constructs through Coding Mobile-Based Games to Secondary School Students. *Informatics in Education* 2020, 19, 543–568, https://doi.org/10.15388/infedu.2020.24.
5. Cheung, J.C.Y., Ngai, G., Chan, S.C.F., Lau, W.W.Y.: Filling the Gap in Programming Instruction: A Text-Enhanced Graphical Programming Environment for Junior High Students. ACM SIGCSE Bulletin 2009, 41, 276–280.
6. Dekhane, S., Xu, X., Tsoi, M.Y.: Mobile App Development to Increase Student Engagement and Problem Solving Skills. *Journal of Information Systems Education* 2013, 24, 299–308.
7. Deng, W., Pi, Z., Lei, W., Zhou, Q.: Zhang, W. Pencil Code Improves Learners' Computational Thinking and Computer Learning Attitude. *Comput Appl Eng Educ* 2020, 28, 90–104, https://doi.org/10.1002 /cae.22177.
8. do Nascimento, M.D., Felix, I.M., Ferreira, B.M., de Souza, L.M., Dantas, D.L., de Oliveira Brandao, L., de Oliveira Brandao, A.: Which Visual Programming Language Best Suits Each School Level? A Look at Alice, IVProg, and Scratch. In Proceedings of the 2019 IEEE World Conference on Engineering Education (EDUNINE); IEEE: Lima, Peru, March 2019; 1–6, https://doi.org/10.1109/EDUNINE.2019.8875788.
9. Garneli, V., Giannakos, M.N., Chorianopoulos, K.: Computing Education in K-12 Schools: A Review of the Literature. In Proceedings of the 2015 IEEE Global Engineering Education Conference (EDUCON); IEEE: Tallinn, Estonia, March 2015; pp. 543–551, http://dx.doi.org/10.1109/educon.2015.7096023.
10. Hromkovič, J., Steffen, B.: Why Teaching Informatics in Schools Is as Important as Teaching Mathematics and Natural Sciences. In Informatics in Schools. Contributing to 21st Century Education; Kalaš, I., Mittermeir, R.T., Eds.; Lecture Notes in Computer Science; Springer Berlin Heidelberg: Berlin, Heidelberg, 2011; Vol. 7013, 21–30, ISBN 978-3-642-24721-7; https://doi.org/10.1007/978-3-642-24722-4_3.
11. Hsu, Y.-C., Ching, Y.-H.: Mobile App Design for Teaching and Learning: Educators' Experiences in an Online Graduate Course. *IRRODL* 2013, 14, 117–139, https://doi.org/10.19173 /irrodl.v14i4.1542.
12. Javidi, G., Sheybani, E.: Teaching Computer Programming through Game Design: A Game-First Approach. *GSTF Journal on Computing* 2014, 4, 1, 17-22.
13. João, P., Nuno, D., Fábio, S.F., Ana, P.: A Cross-Analysis of Block-Based and Visual Programming Apps with Computer Science Student-Teachers. *Education Sciences* 2019, 9, 181, https://doi.org/10.3390/educsci9030181.
14. Koorsse, M., Cilliers, C., Calitz, A.: Programming Assistance Tools to Support the Learning of IT Programming in South African Secondary Schools. *Computers & Education* 2015, 82, 162–178, https://doi.org/10.1016/j.compedu.2014.11.020.
15. Krpan, D., Mladenovic, S., Zaharija, G.: Mediated Transfer from Visual to High-Level Programming Language. In Proceedings of the 2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO); IEEE: Opatija, Croatia, May 2017; 800–805, https://doi.org/10.23919/MIPRO.2017.7973531.
16. Mladenović, M., Boljat, I., Žanko, Ž.: Comparing Loops Misconceptions in Block-Based and Text-Based Programming Languages at the K-12 Level. *Educ Inf Technol* 2018, 23, 1483–1500, https://doi.org/10.1007/s10639-017-9673-3.
17. Musmarra, P.: Reflections on Teaching App Inventor: Challenges and Opportunities. In EC-TEL Practitioner Proceedings 2018: 13th European Conference on Technology Enhanced Learning, 2193, 2018.
18. Noone, M., Mooney, A., Nolan, K.: Hybrid Java: The Creation of a Hybrid Programming Environment. *Irish Journal of Technology Enhanced Learning* 2021, 5, https://doi.org/10.22554/ijtel.v5i1.67.
19. Ouahbi, I., Kaddari, F., Darhmaoui, H., Elachqar, A., Lahmine, S.: Learning Basic Programming Concepts by Creating Games with Scratch Programming Environment. *Procedia - Social and Behavioral Sciences* 2015, 191, 1479–1482, https://doi.org/10.1016/j.sbspro.2015.04.224.
20. Papadakis, S., Orfanakis, V.: Comparing Novice Programing Environments for Use in Secondary Education: App Inventor for Android vs. Alice. *IJTEL* 2018, 10, 44-72, https://doi.org/10.1504/IJTEL.2018.088333.
21. Paternò, F., Santoro, C.: End-User Development for Personalizing Applications, Things, and Robots. *International Journal of Human-Computer Studies* 2019, 131, 120–130, https://doi.org/10.1016/j.ijhcs.2019.06.002.
22. Perkins, D. N., Salomon, G.: Teaching for transfer. Educational Leadership 1988, 22-32.

23. Perkins, D. N., Salomon, G.: Transfer of Learning. International Encyclopedia of Education, Second Edition, Pergamon Press: Oxford, England, 1992.
24. Radosevic, D., Orehovacki, T., Lovrencic, A.: Verificator: Educational Tool for Learning Programming. *Informatics in Education* 2009, 8, 261–280, https://doi.org/10.15388/infedu.2009.16.
25. Saeli, M., Perrenet, J., Jochems, W. M. G., Zwaneveld, B.: Teaching Programming in Secondary School: A Pedagogical Content Knowledge Perspective. *Informatics in Education* 2011, 10, 1, 73–88, https://doi.org/10.15388/infedu.2011.06.
26. Strong, G., O'Carroll, S., Bresnihan, N.: A Block Based Editor for Python. In Proceedings of the Proceedings of the 13th Workshop in Primary and Secondary Computing Education; ACM: Potsdam Germany, October 4 2018; 1–2, https://doi.org/10.1145/3265757.3265788.
27. Tkáčová, Z., Šnajder, Ľ., Guniš, J.: Introducing STEM Activities into Informatics Education through Mobile Apps Development. In ISSEP 2017 – The 10th International Conference on Informatics in Schools, University of Helsinki, Helsinki, Finland, 2017.
28. Tóth, T., Lovászová, G.: On Difficulties with Knowledge Transfer from Visual to Textual Programming. In DIVAI 2018 – The 12th international scientific conference on Distance Learning in Applied Informatics. Conference Proceedings. Wolters Kluwer ČR, a. s., 2018, 379-386.
29. Tóth, T., Michaličková, V.: From App Inventor to Java: A Strategy for Mediating the Transition. In 2018 16th International Conference on Emerging eLearning Technologies and Applications (ICETA), Stary Smokovec, Slovakia, 2018, 591-596, https://doi.org/10.1109/ICETA.2018.8572156.
30. Vega, J., Cañas, J. M.: PyBoKids: An Innovative Python-Based Educational Framework Using Real and Simulated Arduino Robots. *Electronics* 2019, 8, 899, https://doi.org/10.3390/electronics8080899.
31. Weintrop, D.: Minding the Gap between Blocks-Based and Text- Based Programming: Evaluating Introductory Programming Tools. In SIGCSE '15: Proceedings of the 46th ACM Technical Symposium on Computer Science Education 5, http://doi.org/10.1145/2676723.2693622.
32. Weintrop, D., Hansen, A. K., Harlow, D. B., Franklin, D.: Starting from Scratch: Outcomes of Early Computer Science Learning Experiences and Implications for What Comes Next. In Proceedings of the 2018 ACM Conference on International Computing Education Research (ICER '18). Association for Computing Machinery, New York, NY, USA, 2018, 142–150, https://doi.org/10.1145/3230977.3230988.
33. Weintrop, D., Wilensky, U.: Comparing Block-Based and Text-Based Programming in High School Computer Science Classrooms. ACM Trans. *Comput. Educ.* 2017, 18, 1–25, https://doi.org/10.1145/3089799.
34. Weintrop, D., Wilensky, U.: How Block-Based, Text-Based, and Hybrid Block/Text Modalities Shape Novice Programming Practices. *International Journal of Child-Computer Interaction* 2018, 17, 83–92, https://doi.org/10.1016/j.ijcci.2018.04.005.
35. Weintrop, D., Wilensky, U.: Transitioning from Introductory Block-Based and Text-Based Environments to Professional Programming Languages in High School Computer Science Classrooms. *Computers & Education* 2019, 142, 103646, https://doi.org/10.1016/j.compedu.2019.103646.
36. Wing, J. M.: Computational Thinking. *Communications of the ACM* 2006, 49, 3, 33-35, http://doi.org/10.1145/1118178.1118215.

Primary Paper Section: A

Secondary Paper Section: AM, IN