International Journal of Engineering and Management Sciences (IJEMS) Vol. 3. (2018). No. 1.
DOI: 10.21791/IJEMS.2018.1.5.

Tools to develop end-user programming skills from primary school to higher education

VIKTOR LÁSZLÓ TAKÁCS, KATALIN BUBNÓ, RÓBERT SZILÁGYI

University of Debrecen, Institute of Applied Informatics and Logistics, Faculty of Economics and Business, Department of Business Informatics, takacs.viktor@econ.unideb.hu
University of Debrecen, Doctoral School of Mathematical and Computational Sciences, bubno.kati@gmail.com
University of Debrecen, Institute of Applied Informatics and Logistics, Faculty of Economics and Business, Department of Business Informatics, szilagyi.robert@econ.unideb.hu

Abstract. Some years ago at the MIDK2013 Conference [16] we presented a concept regarding the use of a new generation of computer programming novice languages (in fact, an environment, called 'block languages'). We showed how we imagine the use of this kind of language for different ages and at different levels of education. Now we would like to present the results of our projects related to this concept and our psychological and pedagogical experiences. Furthermore we would like to present further tools to develop end-user programming skills from primary school to higher education.

Keywords: end-user development; block languages; programming by example; user programming; graphical programming.

INTRODUCTION

Block programming languages are becoming more and more popular among young and hobby programmers. Some years ago at the MIDK2013 Conference [16] we presented a concept regarding the use of a new generation of computer programming novice languages (in fact, an environment, called 'block languages'). We showed how we imagine the use of this kind of language for different ages and at different levels of education. The aim of this paper is to present the results of our psychological and pedagogical experiences. Furthermore we would like to present further tools to develop end-user programming skills from primary school to higher education.

ABOUT BLOCK PROGRAMMING LANGUAGES

There were several block programming projects available from the 2000s, but the most popular for educational use were Scratch (created and supported by MIT) [13] and LabView (created and supported by National Instruments) [9]. In 2010 a new giant IT company, Google, entered the ring and created a third programming language, which has proved (after Labview) that block programming languages are suitable for developing people's computational thinking at any age or in any sphere of interest. This new programme was App Inventor for Android. Compared to traditional developing environments the technical and syntactical problems were reduced to a minimum, and several hobby programmers began to develop mobile applications for their own devices. Google's timing was perfect: it was the beginning of the mobile world. Before App Inventor only programmers could develop applications for Android mobile devices; now this option became available to laymen as well.

After a few years MIT took over the project, and they began to develop and host App Inventor 2 [8].

However, in the meantime, another avalanche had begun. Thanks to the development policy of Google, they published and shared the soul of the project, i.e. the Blockly programming library [6]. With Blockly we can develop new programming languages (environments) in different domains. In blockly.com several interesting projects are listed. We have created two as well, the first being BlockImpress [17] a tool for web presentation, and the second Blocklino, which encourage Arduino based development (both hardware and software)

DESCRIBING BLOCK LANGUAGES

Describing new developing tools is a difficult challenge. Block languages are mostly end-user programming languages. If we want to evaluate programming languages, we often use Robert Sebesta's criteria [14] which is an excellent tool for the scientific analysis of programming languages and environments. With this criteria we can decide with reasonable certainty whether a programming language is worthy of attention or not. In this section we describe the criteria we use to describe influences on the programming environment, with their advantages and disadvantages.

	Criteria		
Characteristic	Readability	Writeability	Reliability
Simplicity	●	●	●
Orthogonality	●	●	●
Data types	●	●	●
Syntax designs	●	●	●
Support for abstraction		●	●
Expressivity		●	●
Type checking			●
Exception handling			●
Restricted Aliasing			●

Table 1 Language evaluation criteria and the characteristics that affect them.

Simplicity and Orthogonality

Block languages have just a few blocks, because these are designed as easy-to-learn, end-user programming languages. Regarding advanced users, block languages offer both basic and advanced methods, e.g. mutations of blocks or using component collections for each.

Mutation has a special relation to Simplicity and Orthogonality; a block with mutation(s) can be used in two ways. Beginners can use the block without mutation(s) easily, but advanced users can also benefit from the optional parameters of the block.

Component collections can be used in abstract programming when we want to set the width property of every button in a screen, instead of setting the property of the buttons one-by-one, which leads to a higher abstraction level of planning and development.

Data types

Block languages usually work only with basic data types e.g. string, number and Boolean. These types are sometimes expanded with color or any domain specific types, but do not contain any subtype, such as integer or char. This design consideration helps greatly in the beginning of learning a new developing environment, and will not be confusing if the user becomes a professional later on.

Syntax designs

Block languages have a great advantage over common programming languages; we can use different colours for the different block groups, and we can also play with the shapes of the blocks related to their different stacking and embedding types. These methods are very helpful for beginners in developing codes; they have clues as to how to stack the blocks on each other, which is the main element in syntax considerations and can avoid the irritating “syntax error” message. The other advantage of block language syntax consideration is the background type checking method; we cannot stack a string to a plus block, and the user has a real-time visual response, unlike in the common programming languages.

Support for abstraction

Most block languages are intended to support process abstraction, but we can also find some data abstraction support, e.g. colours. Process abstraction is highly important in block languages, and the development method depends on the level of the abstraction. App Inventor 2 contains the Acceleration sensor component which has a shaking event. This event arises based on a relatively complex background computation of three dimensional movement with a Boolean value output.

We can say that a higher level of process abstraction is a great help for the beginner, as is the language, which is simple up to a specific level, since too many abstract processes could be confusing for a beginner, and also orthogonality is a disadvantage.

Expressivity

Most block languages were created to make it convenient for the wider public to use APIs. They offer simple blocks for using sensors, a camera, or a canvas, etc. which also means that the "programmer" does not have full freedom in using these APIs; however, they accelerate the learning process for beginners as they move towards the intermediate level.

Type checking

There is strong type checking to stack blocks on each other in BlockImpress [17].

Exception handling

Exception handling is usually an important part of block languages, but beginners are not really interested in this, although when the program allows the user to input, this is unavoidable. Exception handling is a tool for advanced users.

Restricted Aliasing

This is usually not a relevant problem for block languages.

END-USER PROGRAMMING TOOLS

The main concept of the described end-user programming tool family is problem solving from early ages while we can concentrate on the methods and not on the syntax.

Google Blockly

The first idea was presented in [4], as well. We use it mostly in classrooms, in Computer Science lessons on the subjects of algorithms and computer programming, for teaching basic programming tools as declarations, and making mathematical operations with variables, using loops, conditional statements and simple data structures, such as lists. Our hypotheses were that

- mathematical word problems are suitable for novice programming exercises and,
- with the application of block programming languages as novice languages we can rapidly develop computational abilities and computational thinking [10] between the ages of 12 and 16.

App Inventor 2 (MIT App Inventor)

Mobile programming is always motivating for children and for teachers, too. It became our habit that the first lesson for incoming classes is about developing a rapid mobile application in the Secondary School. There is a concept regarding the economy of attention [7], in the digital age things that do not attract consumers' interest in the first five minutes are probably not viable. In the school we, the teachers, are sellers and our consumers are children. So we have to show them something cool, like mobile programming [18] to arouse their interest in algorithms.

At first sight block languages are similar to Scratch, which was the Massachusetts Institute of Technology Media Lab's project from 2003, and is a very popular self-study programming environment among children all over the world. Scratch is used in more than 150 different countries and is available in more than 40 languages. With Scratch children can program their own interactive stories, games, and animations, and share their creations with others in the online community (scratch.mit.edu). Formerly, Scratch worked as desktop software; nowadays it can be used via the Internet as a cloud service hosted by MIT, similar to other block programming environments.

We have been organising successful study circles and talent development programs in mobile programming with MIT App Inventor since 2013. [2][3]

Although in 2013 we also planned to work with Lego Mindstorm NXT, we realized it was not interesting for children. They are much more interested in mobile programming than controlling robots. Mobile technology was new and trendy, while robots were already familiar to them.

Blocklino

In the next semester our talent program was linked to the "Year of the Light". We introduced Arduino and LED-controlling combined with mobile programming. It was a huge success, surprisingly also among girl users.

There was a new attempt in this field. One of our talented boys, Máté (16), was involved in computer programming in developing the user interface of a new block language, called Blocklino [2]. He worked with a mentor engineer teacher's help, and he surpassed all our previous expectations. Blocklino is a Blockly based programming environment which can control various self-developed LED-based hardware devices. Blocklino and App Inventor are compatible, so students in talent programs can develop applications to control an LED-series from their own mobile phones.

EDUCATION CONCEPT

In [16] we described a concept regarding the use of block programming languages in education.

Educational level	Applications
Primary School	novice programming, basic concepts, solving word problems, making easy applications, web-programming
Secondary School	mobile programming, web-programming, graphical interfaces, graphical programming, introducing serious programming via built-in translators, detecting end-user talents
Higher education (not IT professional education)	developing IT skills (hacking), mobile programming, web-programming, graphical programming, end user development, labor tool in introductory courses
Higher education (business education)	mobile programming, web-programming, graphical programming, end user development, business presentation
Higher education (IT professional education)	DSL programming, code optimizing, cross and multi-platform developing (writing automated translator), user interface programming (ergonomy and design)
Hobby activity	anything from the above

Table 2. Usage of block languages in different educational levels..

Primary school project

In primary school we carried out one project involving mobile programming with App Inventor for Android, beginning in 2012. It involved a study group of children from the ages of 10 to 13. We encountered many problems with the differences in children's knowledge and in their maturity. Mobile computing was very interesting for children. Of the 14 pupils there were 3 boys (Dávid (11), Tamás (10) and Péter (13)) and 1 girl (Vera (10)) who proved to be very talented. It was a normal a primary school class, with no speciality in the subject classes. At that time Tamás and Vera simultaneously learned to program in Scratch by themselves (individually). Two years later both of them were admitted to a six-grade secondary school with a science orientation. Péter, who was already 13 years old, was older than the others. He chose a secondary technical school to develop his education in the following year. Dávid was the most important experience for us, because he had learning difficulties, he was somewhat dyslexic, and with each subject he needed more time to learn or solve exercises. But now, with bock languages he can work not just at the same pace as the others, but even more rapidly than them, and he had a great sense of achievement because of this.

So, we think with the App Inventor study group we have found four talented children. We can state that a motivating block language can be suitable for detecting talented children in informatics as early as the primary school.

Secondary School projects

Since 2013 at the Kossuth Grammar School of the University of Debrecen we have been trying out more than one block programming language in the classroom, both in study groups and in talent programs. We solved mathematical word problems with Blockly Code and made several android application with App Inventor 2.

Higher Education projects

Since 2014 at the University of Debrecen Faculty of Economics and Business we had several student thesis in the field of android application development via Appinventor. The students have various programming skills, they are mainly practice oriented persons. They prefer App Inventor2 because they imagined that applications can be develop faster and required less programming skills. In every week students has a special course where they can ask teachers for the actual development step.

In the Faculty of Economics and Business the following applications created by students in the last semesters:

- „How the bread made” Multimedia mobile learning materials for children,
- AgrInfo Android land information application
- FarmGPS mobile application for farmers

Instead of benefits the App Inventor2 has some limitations. The complex applications required to use another solutions. The AI2 was popular because most of the students has some software development background, but the application development were less popular earlier.

End User Development in higher business education

We have two type C courses and an informal talent program.

On the Mobile Technologies in Business course, we teach some basics about the technology and development of mobile applications with App Inventor 2, such as a QR-code based shopping app, a simple ordering app, and an informational app.

On the Web technologies in Business course, we teach some basic web technologies and create web based presentations with BlockImpress.

In our talent program we are working on various topics related to planning and developing databases, analysing and visualizing data from a data warehouse, and creating mobile apps or web presentations. We organise an Android Teahouse once a semester, when we show our work to the public and involve new participants in our program.

End-Users and the Sebesta criteria

The easy readability and writeability of codes are key factors if we want to create end-user coding in an exploratory and creative way. It most cases it is not the best looking code, and it is not even the best performing code, but it gets the job done. The reliability of the code is often not important for end-users, although fortunately, most block languages contain reliable code behind the blocks. End-users programmers' coding skills that others mention as hacking skills [12] is about the ability to break down messy problems and recompose them in ways that are solvable. It is nothing more than discussion, the last and highest step in Polya's problem solving method [11] when applied to information technology tools.

We are convinced that end-users can be developed from early childhood through to adulthood with teaching domain specific visual programming environments, like App Inventor and the others described above.

We are also convinced that developing this skill is not only useful in business education but in many other professions where informatics needs to be applied.

Mental transformation

What we miss in the literature on the education of Business Informatics is the ability referred to in psychology as mental transformation. Mental transformation is one of the five elements of spatial intelligence. Bakó [1] in already studied, how a computer program could help to improve this ability. We think developing this skill is as important as developing end-user coding skills. Developing mental transformation occurs mostly in geometry, for example when primary school children already need to imagine whether or not a planar lattice could be a lattice of a 3D solid. When business students encounter matrices and accompanying transformations on a university mathematics course, they hardly (or never) recognize the connection. And they hardly ever, or never, recognize the connection in their work with Excel. It would be very useful to develop the mental transformation skill because they support the teaching of business intelligence, as well as monitoring and analytical skills.

We found a good tool to support this skill – Sprego programming. Sprego is a method for teaching spreedsheets which is more conscious than other methods, which mostly teach Excel (or other spreedsheets) only technically. In [5] we read about the important result, i.e. that knowledge about spreadsheets can be sustained for a much longer period when we teach it with the Sprego-method.

Sprego is a non-traditional functional language and is modular, so it also fits with our concept of developing coding skills. It is also based on the development of the mental transformation skill, and after presenting it in introductory courses we can continue pivoting, reporting and using later self-service business informatics tools, such as Excel Power Pivot, on which we have reported in [15]; it can also help us develop macros and introduce script languages.

DOI: 10.21791/IJEMS.2018.1.5.

BENEFITS OF THE METHOD

We must strengthen self-sufficient problem solving capabilities, instead of copying from tables. Moreover, the future challenge - and the growing expectation on the part of society - is to realize these abilities in a computer science classroom environment, rather than in the traditional paper based environment.

Exercises involving different kinds of sorting or examining anagrams are extremely boring for today's children, and do not seem useful for them at all. Consequently, there is a big problem with students' motivation in computer science lessons.

Traditional computer science education already teaches programming languages in primary school. In Hungarian curricula teachers have a lot of freedom to choose any kind of programming language they want to teach within the subject of algorithms and programming, including the option to teach only block diagrams to make algorithms, and nothing else. And, to tell the truth, the lessons available for any discussion of this topic hardly allow anything else. Block languages facilitate teachers in teaching not just a language, but a family of languages, including different types of programming methodologies, ranging from sequential to event-driven methods. We can show students much more about the diversity of programming skills and computational thinking.

Blockly has a translator for many other programming languages, so pupils seriously interested in computer programming can also be introduced to "serious" computer programming. We have given them a "vocabulary" to study a language by themselves.

FUTURE QUESTIONS

We are very committed to teaching this new technology, but to tell the truth, we also have doubts and many questions for the future.

Is teaching a family of programming languages (or related programming environments) instead of 1-2 concrete languages which are not used in computer science today a good direction to follow in computer science education or not? It is modern and can present a fuller picture of today's computer science, but is it not too difficult for children?

Sorting is boring. But how long will mobile programming be interesting and motivating for children: for 5 or at most 10 years? And after that?

The future direction of technology is unpredictable. Can an information technology teacher follow it? Is it a realistic expectation that technology teachers can follow and teach technologies which are always new? How can we prepare the next teacher generation for this challenge?

ACKNOWLEDGEMENTS

Our talent programs in the Kossuth Grammar School of the University of Debrecen was supported by the NTP-MTI-13-0049 and NTP-MTI-MPA-14- 0099 National Talent Programs. Katalin Bubno was supported by TÁMOP-4.2.2.D-15/1/KONV-2015-0002 program.

References

[1] Bakó, M. (2004): Rotation mentale et ordinateur. Teaching Mathematics and Computer Science 2(1), 33-48.

[2] Bécsi, Z., Bojda, B., Bubnó, K., Domokos, P., Kelemenné Nagy, A., Takács, V.L. (2015): *Mobile programming by block language.* (In Hungarian). In Stettner, E., Klingné Takács, A., Barna, R. (Eds.) Proceedings of the 39th MAFIOK Conference, Kaposvár, 2015. augusztus 24-26, (pp. 83-88). Kaposvár, Hungary: University of Kaposvár.

[3] Bécsi, Z., Bojda, B., Bubnó, K., Kelemenné Nagy, A., Takács, V.(2016). *Teaching block programming languages* (In Hungarian). In Proceedings of Tudóstanárok és tanár tudósok Conference. Budapest, Hungary: Eötvös Loránd University. (in press)

[4] Bubnó, K., Takács, V. L. (2014). *Solving word problems by computer programming*. In András Ambrus, Éva Vásárhelyi (Eds.), Problem Solving in Mathematics Education, Proceedings of the 15th ProMath Conference 30 August-1 September, 2013 in Eger. (pp. 193-208). Budapest, Hungary: Eötvös Loránd University.

[5] Csernoch, M., Biró, P., Máth, J., Abari, K. (2015). *Testing Algorithmic Skills in Traditional and Non-Traditional Programming Environments*. Informatics In Education 14(2), 175-197.

[6] Faser, N. (2012). *Google Blockly*. (Retrieved at 20 December, 2015 from https://developers.google.com/blockly/)

[7] Hámori, B., Szabó, K. (2005). *Information economy* (In Hungarian). Budapest, Hungary: Akadémiai Kiadó.

[8] Massachussetts Institute of Technology (2012). *MIT App Inventor*. (Retrieved at 20 December, 2015 from http://appinventor.mit.edu/)

[9] National Instrument Hungary (2015). *LabView 2015*. (Retrieved at 20 December, 2015 from http://hungary.ni.com/labview)

[10] Papert, S. A. (1993). *Mindstorms. 2nd ed*., New York, NY: Basic Books.

[11] Polya, G. (1957). *How to solve it?* 2nd ed. Garden City, N.Y. : Doubleday

[12] Press, G. (2013). *A Very Short History of Data Science*. (Retrieved at 20 December, 2015 from http://www.forbes.com/sites/gilpress/2013/05/28/a-very-short-history-of-data-science/#4d98b42f69fd5d24a5a569fd)

[13] Resnick, M. (2003) *Scratch*. (Retrieved at 20 December, 2015 from https://scratch.mit.edu)

[14] Sebesta, R. (2010). *Concepts of programming languages*. 9th ed. Upper Saddle River, NJ : Pearson.

[15] Takács, V. L., Bubnó, K. (2012). *Educational experiments of self-service business intelligens tools*. (In Hungarian). Acta Carolus Robertus 3(1), 99-104., 2013.

[16] Takács, V. L. (2013). *Block programming languages in education* (in Hungarian). Oradea, Romania, MIDK2013 Conference. (Retrieved at 20 December, 2015 from gmt.partium.ro/103729/midk2013-016.ppt)

[17] Takács, V. L. (2014). *BlockImpress*. In Proceedings of the 18th International Conference on Intelligent Engineering Systems (INES), 2014. (pp. 221-226). IEEE Computer Society.

[18] Wolber, D., Abelson, H., Spertus, E., Looney, L. (2011). *App Inventor*. Sebastopol, CA: O'Reilly.

[19] Wolber, D., Abelson, H., Spertus, E., Looney, L. (2014). *App Inventor 2*. Sebastopol, CA: O'Reilly.

[20] Zhou, Sharon, Ivy J. Livingston, Mark Schiefsky, Stuart M. Shieber, and Krzysztof Z. Gajos. Ingenium: *Engaging Novice Students with Latin Grammar*. Proceedings of the 34th Annual ACM Conference on Human Factors in Computing Systems (CHI '16), San Jose, CA, May 7-12, 2016.