Turtles, Robots, Sheep, Cats, Languages what is next to teach programming? A future developer's crisis?

F. Edgar Castillo-Barrera[1], **P. David Arjona-Villicana**[1], **Cesar A. Ramirez-Gamez**[1], **F. Eloy Hernandez-Castro**[1], **and S. Masoud Sadjadi**[2]
[1]School of Engineering, Universidad Autónoma de San Luis Potosí, San Luis Potosí, México
[2]School of Computing and Information Sciences, Florida International University (FIU), Miami, USA

Abstract

This paper narrates the process that our Computer Science department has followed to drastically change the introductory programming courses. Traditionally, our students were learning to program using the C language, but because of the high failure rate it was decided to first teach basic programming skills using two programming support tools: Raptor and Scratch. Although the passing rate has increased, it has not been possible to determine if this new teaching strategy has eased our students to learn programming in **C** *or* **C++**. *We have also noticed that Raptor does not offer all the basic features required in the syllabus of our department, therefore an analysis of other programming support tools which may offer a richer set of characteristics is provided.*

1 Introduction

Programming courses and curricula have been evolving since they started and there seems to be no end to this process. The **BASIC** language was very popular as a teaching resource, but it lacked support for structured programming and a modular structure. Hence, Pascal became an answer to Basic's limitations. Nowadays, the most popular languages are **C**, **C++** and **Java**[7]. Our department is no different and until the fall of 2011, it used to introduce C as the first programming language. This produced a very high failure rate for the Introduction to Programming course.

Other programming tools offer the concept of a microworld, where a character performs actions based on instructions and within the environment or world defined by its creators. The first tools that used this concept were **Logo** turtle graphics and **Karel** the robot. However over time, it's very simple animated GUI ceased to be interesting to many students because technology provided better graphics and interfaces.

Still the main problem persists: most students do not develop a proper programming logic and cannot correctly employ control structures for programming [9]. New programming support tools with multimedia, like **Scratch** [20], have recently emerged. Unfortunately, Scratch introduces the students to programming based on the events paradigm, while the teaching objectives we seek to develop at our first programming course follow the linear and procedural paradigm.

We think that it is necessary to document the process we have followed to increase the passing rate of our students. This has required that we analyze different programming support tools and employ them in an introductory course. On the other hand, this process has not concluded and we will continue learning from these experiences in order to increase the chances of success for our next curriculum changes.

This paper is structured as follows: Section 2 shows the related work. Section 3 provides a description and analysis of the tools used at our department. Section 4 describes the experience using programming languages. Finally, Section 5 provides a discussion and conclusions of this work.

2 Related work

The literature review on how to teach programming skills to university students concurs that learning to program is difficult [12, 17]. Therefore, there have been many proposals for strategies and tools which could facilitate the learning process of this skill. One of such strategies has been to implement a pre CS1 course (CS0) in which students are familiarized with algorithmic and problem solving skills which could later be applied when trying to learn a standard programming language [8, 14], like **C**, **C++** or **Java**.

Computer science courses which try to develop programming skills without using a standard language usually employ a programming support tool. Most of these tools provide an easy mechanism to implement a program or algo-

Table 1. Analysis of tools used for learning and teaching programming

Tool	Subroutines	Parameters	Animated	Flowchart	Multimedia	Debugger	Statements for Graphics
KAREL	Yes	No	No	No	No	No	No
DFD	Yes	Yes	No	Yes	No	No	No
Raptor	Yes	Yes	No	Yes	No	No	Yes
Logo	Yes	No	No	No	No	No	Yes
Scratch	Yes	No	Yes	No	Yes	No	No
Byob	Yes	Yes	Yes	Yes	Yes	Yes	No
Jeroo	Yes	Yes	Yes	No	Yes	No	No
Greenfoot	Yes	Yes	Yes	No	Yes	Yes	No
PSInt	Yes	Yes	No	Yes	No	Yes	No
Alice	Yes	Yes	Yes	No	Yes	Yes	Yes

Table 2. Analysis of tools used for learning and teaching programming

Tool	Linear Arrays	Bidimensional Arrays	Variables	if	while	do-while	for	repeat until
KAREL	No	No	No	Yes	No	No	No	Yes
DFD	No	No	Yes	Yes	Yes	No	Yes	Yes
Raptor	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes
Logo	Yes	No	Yes	Yes	Yes	No	No	Yes
Scratch	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes
Byob	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes
Jeroo	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes
Greenfoot	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No
PSInt	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes
Alice	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No

rithm and a visual feedback that allows students to see the effects to the changes they make. In a SIGCSE panel session [13] the following tool classification was proposed:

- Narrative tools: help to create a customized story which in turn increases the student motivation and engagement, e.g. **Alice** and **Jeroo**.
- Visual programming tools: allow students to drag self-contained units of code and create their own programs without having to worry about the program syntax and coding errors, e.g. **Alice**, **Greenfoot** and **Karel**.
- Flow-model tools: implement a flow-model which shows student how a program should work and bypasses the need to write code e.g. **Raptor** and Iconic Programmer.
- Specialized realization: like **Lego Mindstorms**, allow students to physically experience the results of their implementations.

It is important to note that none of these tools will actually teach how to create a program in a standard language, like C or Java. Therefore, there is a gap between tools and languages. On the other hand, these tools encourage the development of the skills needed to learn a standard programming language. This means that, at least in theory, students that have successfully completed their CS0 course should be half-way to become novice programmers. Therefore, in order to evaluate the success of these tools it is necessary to wait until students have finished their equivalent CS1 course to evaluate if they have really learned to program.

Although there have been efforts to develop a standardized test to evaluate the programming proficiency of students [9], we are not aware if this test has been widely accepted by the academic community.

3 Tools

The use of tools with visual characters and animated objects using a language based on blocks are now more popular in the first course of programming. In fact, languages such as **BASIC** and **PASCAL** are no longer common in the first programming courses.

3.1 Using a simple character: Logo and Karel

Logo was created in 1967 at Bolt, Beranek and Newman (BBN). A Cambridge, Massachusetts research firm created by Wally Feurzeig and Seymour Papert [18]. Although successes have been reported in the use of microworld programming tools, the work done by Xinogalos [21] suggests the opposite. After Logo, the most used tool to teach programming in universities for first-year students was **Karel** the Robot, developed by Richard E.Pattis [11] [16] [19][1]. Another version of Karel, which focuses on Object Oriented Programming, was made by Xinogalos [22] and is called **objectKarel**. Based on the experience of other universities and the literature reviewed [17], we have decided not to start teaching to program using the object-oriented paradigm.

3.2 Flowcharts: Raptor and DFD

DFD was developed by the Smart group at University of Magdalena, Colombia [3]. This tool allows you to create flow charts with subroutine calls and execute them step by step. An even better tool also based on flow charts is **Raptor** [4], which incorporates the use of arrays, graphical mode, and a debugger. It also allows to create class diagrams in UML.

3.3 PSeInt: Pseudocode, Flowchart and more

PSeInt is a tool to learn the logic of computer programming based on pseudocode. Through the use of a simple and limited pseudocode-language and an intuitive user interface in Spanish. The students can begin to understand basic concepts of control structures, supported by a modular and a procedural programming paradigm. In addition, this tools allows the use of flowcharts, functions and procedures with parameters, and multidimensional arrays are also supported.

Although in our evaluation of tools PSeInt ranked as one of the best, it still lacks some desirable features (see Tables 1 and 2): It is only available in Spanish and does not support multimedia or animated features, which are attractive to students.

[1]A new version of Karel, called **Critters** [2], has been developed by Anderson and McLoughlin.

3.4 Multimedia: Scratch, Byop, Alice and Greenfoot

Visual programming tools, such as **Scratch**, **Byop**, **Alice**, **Greenfoot**, have been developed to attract children and young people to learn programming. They usually allow the user to develop their own games and provide friendly interfaces.

3.4.1 Scratch

Scratch [15] is defined as a programming language by its creators. It was developed at MIT by the Lifelong Kindergarten Group. Scratch allows building games, simple apps and animations, using blocks classified by their functions with colors, therefore the programmer does not have to learn commands, keywords or a specific syntax. Therefore this is a tool which tries to develop skills to program in an Object Oriented Language. Scratch is intended for 8 to 16 years-olds but it is being used in introductory computer science classes at some universities [15]. We selected Scratch as one of the tools for teaching programming because its graphical interface motivates our students to practice and develop their own applications.

The topics cover in our first programming course are selection statement, loops, functions and one-dimensional and two-dimensional arrays. We have been working with this tool for four terms, and in the paragraphs below we describe the advantages and disadvantages we have found.

Scratch has two different selection statements: the *if* statement and the *if - else* statement. Their logic is the same as C language, and nested selection statements are allowed. However, there is not a block to represent the switch command or multiple selections.

Scratch implements four blocks that represent loops, two of them are infinite loops, there is a while loop with a conditional at the beginning, and there is a structure similar to a *for* loop: it occurs a specific number of times but it does not have a variable to control the iterations. Finally, there is no block to represent the *do-while* loop.

One of the biggest problems using Scratch is the absence of functions, procedures or subroutines. Scratch works based on events and objects, therefore Scratch is not a sequential programming language such as C. Scratch allows sending messages from one to several objects.

Scratch has lists insted of arrays, which are different. Lists can have elements of different types and there are blocks that represent functions like orderly insertion, deletion of elements in a specific position, addition of elements in a specific position and replacement of elements. Those functions are not predefined for arrays in languages such as C. Two-dimensional arrays do not exist, neither the option to define list of lists. There are other points that cause confusion at the moment of learning how to program in Scratch.

- The program can have many entry points, which means a program can start with an object, or two at the same time, or maybe the same object with two different actions.
- Scratch support threads that are a great programming tool, but having access of these when you start programming makes the learning process more complex.
- There are few functions and operators, e.g. the $>=$ and $<=$ operators do not exist.
- There are not rules for naming at the moment to create a variable or a message.

On the other hand, **Scratch** is pleasing for students and teachers because:

- Creating animations is fast and easy.
- Allows to import images and sounds.
- It allows to develop creativity.
- There is a web site where students, teachers and developers can find material, share ideas and projects.
- There are also some complements for Scratch, such as the PicoBoard which is a device with sensors that allow the computer to interact with variables in the external environment.

3.4.2 Byob

Byob (Build Your Own Blocks) is an extension of Scratch. We have not used it in class yet, but it provides the opportunity to solve three important issues found in Scratch.

- Provides the ability to create blocks by the programmer, which means the student can implement functions. In fact, these blocks can receive parameters. The problem with these blocks is that the external variables are recognized even though they are not declared inside the block.
- Multidimensional lists are supported, but programming them using blocks is not an easy task.
- Byob has two extra blocks for managing ASCII code, which allows the use of strings.

Some disadvantages of Scratch can be overcome by Byob, but first it needs to correct some details and increase the existing documentation.

3.4.3 Alice

Alice seeks to teach programming by employing a 3D graphical interface. The objective for students is to program and animate 3-D objects (e.g. people, animals, and vehicles) in a virtual world. Alice is a programming environment designed to teach computer programming to children. It uses a similar approach to Scratch by dragging and dropping instructions without requiring a text editor, but it differs in that you can see the generated code in Java. In addition to the basic language components such as variables, arrays, loops and conditionals, it also allows the use of functions and parameter passing. Although it is based on **Java**, it is not required to know the concepts of object-oriented paradigm for its use.

Because Alice is based on Java, and since the objective of our department is to teach our students to use the **C** language, we have decided not to use Alice in our courses. Because there might be a transition problem from Java to C.

3.4.4 Greenfoot

In the concept of microworlds, **Greenfoot** is an environment for teaching object oriented programming skills. It employs the **Java** language and it is a simplified programming environment for developing Java applications. It has a set of classes that enable the development of programs directly without starting from scratch. In addition, all the Java language features such as variables, loops, conditionals, classes and objects are available. Although **Greenfoot** is designed to be a programming environment for beginners, we use this tool to teach Object Oriented Programming in our later programming courses. In our experience **Greenfoot** has proven to be successful for this pourpose. In fact, it is designed to be used as a programming environment for beginners. There is a tendency to use Scratch or Alice afterwards.

4 Languages: PASCAL, C, C++ and Java

C, **C++** and **Java** are still the more popular languages in the software industry [7], which is why most universities employ these languages for their introduction to programming courses. Our experience have shown us that the use of C as a first language to learn how to program is difficult for our students.

4.1 PASCAL

As a result of the deficiencies of **BASIC** (the most popular language in the mid-1970s), **PASCAL** was considered by many to be a good option as a first language to learn programming. **PASCAL** is an elegant and complete language under the structured paradigm. However concerns were identified when developing the skills of programming logic. This led to the conclusion that it was necessary to develop this skill before learning the language. Dr. Richard Patys developed Karel robot language, associated with the concept of microworlds [21]. But based on our analysis of the concepts needed to learn the student **KAREL** found that did not meet all the requirements, so we proceeded to look for other programs. Moreover its graphical interface is no longer attractive to students with the development of graphics and multimedia.

4.2 C

The first language used in our Department, for the bachelor of Informatic and Computer Engineering, was PASCAL. After that the academic staff decided to use the **C** language for their first programming course. It is considered the most complete language for programming in a structured way and with other characteristics that allow its use in other subsequent courses (operating systems, compilers, graphs, numerical analysis, etc.)

In previous years, we have tried to improve our *Introduction to Programming with C* course by employing structures that would allow to define graphical and three-dimensional objects. We believed that this structures would develop the logical skills needed for programming, and at the same time provide an interesting environment for our students. Moreover, we would encourage students to develop their own video games. Unfortunately, the percentage of failure among these students was still very high. This led us to create a previous course where the only focus is on learning the concepts of structured and modular programming, looking for programming activities that are simple and attractive to students. We have found that it is very difficult to find a tool that includes all the loops in the C language: *while*, *do-while* and *for*.

A tool called C-Sheep, which is based on C and Karel and was developed by Anderson and McLoughlin [1], is another option that we are going to analyse for its possible use.

4.3 C++ and JAVA

For years it has been discussed whether to teach programming using an object-oriented language like **C++** or **Java**. A more complete discussion and review was made by Robins et al. [17]. Based on the experience reported from several universities and considering that the foundation of the object-oriented paradigm is structured programming, and also to the fact that this paradigm seeks to facilitate the programming of complex problems which is not the focus of instruction for a first programming course, we have concluded that it should not be the language for teaching an introductory course to programming.

5 Discussion and Conclusions

Dijkstra [5] suggested the use of mathematics as a way to teach programming, but our experience has shown low interest in young people to learn math, so it is not considerated a good choice.

Based on the experience gained and the consulted literature, we have not found definite proofs that using a visual tool, such as Scratch, is better than using a scripting language, a language of diagrams such as Raptor, or a programming language such as C. However, based on our analysis, we suggest that a tool should be created with the characteristics mentioned in Table 2, or to complement the PSEint tool. However in the case of PSEint, it might be better to be translated into English, to add further control structures required in our syllabus, and also it might be necessary to add a lively visual atmosphere such as Scratch that could motivate students. With these changes we believe that the number of failed students might decrease. Nevertheless, we need to evaluate in a later course if students are really learning how to program.

Students without experience in programming computers require tools that motivate them to learn and to acquire programming skills. In this era of technology, programming applications for cell phones, social networks, chats, web pages, tablets and video games are sometimes the most important concerns of young people.

This article has made the analysis of the results obtained using software tools to teach a structured programming as first course in Introduction to Computer Programming. In past years the course taught using the **C** programming language had a passing rate of around 30%. Once we employed tools based on **microworlds** and **flowcharts** the statistics showed an increase in the passing rate of around 20%. However we believe it is necessary to build a tool with the features mentioned above in order to furhter increase the passing rate, and that could lead to a proper learning of computer programming. We want to emphasize that passing a course of introduction to programming using a tool does not guarantee that the student knows how to program and it is not enough to acquire the required programming skills. In our second programming course based on C language, only 18% passed the course in December 2012.

Students failing in introductory programming courses are also more likely to drop their degree altogether. This, together with the fact that there is a decrease in applications to join in a bachelor of Computer Engineering [6][10], makes the problem of teaching programming even more important. With advances in modern technology, students demand programming tools for learning, in which they can apply to their interests, e.g. social networks, cell phones applications, web pages and video games [2]. Thus these applications could be used in introductory programming courses as a way to motivate and introduce students to the core computer science courses.

6 Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant No. OISE-0730065.

References

[1] E. F. Anderson and L. McLoughlin. C-sheep: Controlling entities in a 3d virtual world as a tool for computer science education. 2006.

[2] E. F. Anderson and L. McLoughlin. Do robots dream of virtual sheep: Rediscovering the" karel the robot" paradigm for the" plug&play generation". 2006.

[3] C. N. Cardenas, F. and D. Eduardo. Universidad del magdalena, santa marta, colombia. Grupo Smart, 1998.

[4] M. C. Carlisle, T. A. Wilson, J. W. Humphries, and S. M. Hadfield. Raptor: a visual programming environment for teaching algorithmic problem solving. In *ACM SIGCSE Bulletin*, volume 37, pages 176–180. ACM, 2005.

[5] E. W. Dijkstra et al. On the cruelty of really teaching computing science. *Communications of the ACM*, 32(12):1398–1404, 1989.

[6] S. Guia, S. Campus, and S. Talento. Fomentar el ingreso y la permanencia en carreras de ti.

[7] R. S. King. The top 10 programming languages. Technical report, IEEE Spectrum.

[8] M. Klassen. Visual approach for teaching programming concepts. In *9th International Conference on Engineering Education*. INEER, Jul 2006.

[9] M. McCracken, V. Almstrum, D. Diaz, M. Guzdial, D. Hagan, Y. B.-D. Kolikant, C. Laxer, L. Thomas, I. Utting, and T. Wilusz. A multi-national, multi-institutional study of assessment of programming skills of first-year cs students. *SIGCSE Bulletin*, 33(4):125–180, Dec 2001.

[10] B. Parhami. Puzzling problems in computer engineering. *Computer*, 42(3):26–29, 2009.

[11] R. E. Pattis. *Karel the Robot: A Gentle Introduction to the Art of Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1981.

[12] A. Pears, S. Seidman, L. Malmi, L. Mannila, E. Adams, J. Bennedsen, M. Devlin, and J. Paterson. A survey of literature on the teaching of introductory programming. *SIGCSE Bulletin*, 39(4):204–223, Dec 2007.

[13] K. Powers, P. Gross, S. Cooper, M. McNally, K. J. Goldman, V. Proulx, and M. Carlisle. Tools for teaching introductory programming: What works? In *Proceedings of the 37th SIGCSE technical symposium on computer science education*, pages 560–561, Mar 2006. This was a panel session, not really an article.

[14] D. Reed. Rethinking cs0 with javascript. *SIGCSE Bulletin*, 33(1):100–104, Feb 2001.

[15] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman, et al. Scratch: programming for all. *Communications of the ACM*, 52(11):60–67, 2009.

[16] E. Roberts. Karel the robot learns java. *Department of Computer Science Stanford University*, 2005.

[17] A. Robins, J. Rountree, and N. Rountree. Learning and teaching programming: A review and discussion. *Computer Science Education*, 13(2):137–172, 2003.

[18] C. J. Solomon and S. Papert. A case study of a young child doing turtle graphics in logo. In *Proceedings of the June 7-10, 1976, national computer conference and exposition*, pages 1049–1056. ACM, 1976.

[19] R. H. Untch. Teaching programming using the karel the robot paradigm realized with a conventional language. *On-line at: http://www. mtsu. edu/untch/karel/karel90. pdf*, 1990.

[20] I. Utting, S. Cooper, M. Kölling, J. Maloney, and M. Resnick. Alice, greenfoot, and scratch–a discussion. *ACM Transactions on Computing Education (TOCE)*, 10(4):17, 2010.

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

[22] S. Xinogalos, M. Satratzemi, and V. Dagdilelis. An objects-first approach to teaching object orientation based on objectkarel. In *Proceedings of the 5th WSEAS International Conference on Education and Educational Technology (EDU'06)*, page 93, 2007.