Visual Programming Languages for Programmers with Dyslexia: an Experiment

José L. Fuertes
*Departamento de Lenguajes y Sistemas
Informáticos e Ingeniería de Software
Universidad Politécnica de Madrid
Madrid, Spain
jfuertes@fi.upm.es*

Luis F. González
*Universidad Politécnica de Madrid
Politécnico Colombiano Jaime Isaza Cadavid
Medellín, Colombia
lfgonzaleza@elpoli.edu.co*

Loïc Martínez
*Departamento de Lenguajes y Sistemas
Informáticos e Ingeniería de Software
Universidad Politécnica de Madrid
Madrid, Spain
loic@fi.upm.es*

Abstract—A survey of 351 programmers, including people with dyslexia, has found that programmers with dyslexia are 33.4% more productive if they use a visual programming language. This paper presents important aspects of the profile of programmers with dyslexia, introduces the most commonly used textual programming languages (TPL) by a group of programmers, classifies the 31 most common visual programming languages (VPL) for the study group, and analyzes some technical and facilitation features to support the needs of programmers with dyslexia, in five of these languages that have been considered current and relevant for the purposes of this research. The visual programming language Alice has been selected as the language to be included in a comparison experiment with the Java programming language. Results of the experiment establish preferences and levels of effectiveness of Alice against the other language, according to the performance of a group of programmers who participated in the experiment.

I. INTRODUCTION

Studies conducted by Rello and Baeza-Yates [1] claim that the percentage of the population admitting to having dyslexia has grown considerably since Steve Jobs publicly confessed that he suffered from the effects of this learning disorder. In fact, many people who had not been diagnosed as having the disability or were not receiving treatment for the disorder have come forward, and some have even sought professional help. However, dyslexia in adults tends to be disregarded, because they learn to live with the disorder and break off professional treatment, without realizing that this functional diversity can affect their communication with other people or their job performance.

Studies in computer science, and particularly programming, have found that dyslexia has an impact and has been a barrier limiting the efficiency of software development processes. As far as we know, there has been no other research to date focusing on specifically helping computer programmers with dyslexia. However, there are studies [2] corroborating the wide-ranging problems that people with dyslexia have with computer use, concluding that students from this population group working with programming tools with visual or graphic aids are more successful at developing and understanding computer programs than programmers without dyslexia.

There are statistical data to show that 12.4% of a group of computer programmers state that they believe that they have dyslexia [3]. On this ground, one member of a team of programmers may very well have symptoms of dyslexia. This may have a bearing on the expected results of the group. However, there is no proposal to help programmers with and without dyslexia to interact in a team development environment.

This paper is part of a PhD research project whose goal is to define an interaction model for computer programmers that would be beneficial for programmers with dyslexia and that would enable them to participate in development teams. This model would be an important contribution to this area, making a positive impact on software development. This would result in greater efficiency and reduced development times for a team of programmers with dyslexic members.

To design the proposed model, different tasks have been carried out which are presented in this work and which were necessary to identify the visual programming language with which programmers with dyslexia interact best.

This paper starts by giving an account of the major emotional, behavioral, physical and programming-related characteristics identified in programmers with dyslexia during software development [4]. It also establishes the relationship of these characteristics with specific VPL elements. It then reports a preliminary classification of the VPLs that the programmers that took the survey identified as being the most popular according to their purpose, relevance and applicability. It then describes an analysis of the main technical and other facilitation features for programmers with dyslexia, of the five languages identified as possible benchmarks for the selection of the ideal VPL to include in a comparison experiment between programmers with and without dyslexia, to measure the level of satisfaction and efficiency in the use of a visual programming language versus a textual programming language and justify the design of the proposed model. Finally, the details and results of the experiment are presented.

II. MATERIALS AND METHODS

This research started with a review of the existing scientific literature to identify the characteristics of adults with dyslexia. Based on the identified characteristics we developed a quantitative cross-sectional study in 351 computer

programmers, chosen using double sampling (at convenience and discretionary snowball), to obtain a map of needs, skills and characteristics of programmers with dyslexia. This allowed us to define the profile of programmers with dyslexia and to determine their specific needs to work with a language programming. Some results of the previous study, together with the literature review, allowed the identification of the 31 visual programming languages most used by the respondents. These languages were classified according to their purpose, validity and relevance, resulting in 5 languages to be included in future studies. With these, we carry out a descriptive analysis of the technical characteristics and another observational analysis of some nontechnical characteristics related to the profile defined in the previous point was carried out, which can be considered as facilitators for programmers with dyslexia. The study allowed for the establishment of a VPL that in the future could become ideal to help minimize the effects of dyslexia in a group of programmers working as a team.

In a third phase, a classification and analysis of the most commonly used textual programming languages by the group of programmers was performed. Object and frequency of use were analyzed, and this led to the selection of Java as the language to be included in future experiments.

In the final phase, an experimental observational study was carried out on the use of Java and Alice programming languages. 39 computer programmers participated, 15 of whom claimed to have dyslexia symptoms.

III. ANALYSIS OF PROGRAMMERS WITH DYSLLEXIA AND PROGRAMMING LANGUAGES

A. Profile of Programmers with Dyslexia

An information gathering method was applied to a sample of 315 Spanish-speaking programmers [4]. 12.4% of all respondents believed that they are dyslexic (n= 39). The above survey researched variables divided into three groups:

- Demographic variables. The following variables were analyzed in this area: sex, age, and nationality.
- Behavioral variables. The following groups of variables were analyzed in this area: emotions, behavior, and physical traits.
- Programming variables. The following variables were analyzed in this area: usability problems, preferences for different programming languages.

The most recurrent characteristics among programmers with dyslexia were identified and they were found to make fewer mistakes when working with a visual programming language. In addition, 33.4% were better able to develop and understand visual programs than programmers that did not have symptoms of dyslexia, thereby confirming the hypothesis stated by Dixon [2]. By synthesizing the characteristics identified by the study within this population group, it was concluded that not all the emotional, behavioral and physical traits of adults with dyslexia reported by other researchers [5] had a bearing on the definition of the profile of the programmer

with dyslexia. However, it was established which characteristics play a role in defining this profile[6].

1) *Behavioral variables*: The results yielded by previous research reveal that programmers with dyslexia have different emotional and behavioral traits than other programmers. The most significant are related to:

- Anxiety: 40.4% of programmers with dyslexia suffer from higher levels of anxiety when programming.
- Concentration: programmer concentration rates are generally high when programming.
- Distraction: despite the above finding, programmers with dyslexia tend to get easily distracted when studying.
- Depression: rates of depression do not tend to be high among programmers with dyslexia.
- Retiring: unlike other people with dyslexia, programmers with dyslexia are not withdrawn and are able to easily socialize with other people.
- Loss of focus: programmers with dyslexia may be easily sidetracked if there is too much information on screen when they are performing programming tasks.
- Inconspicuousness: programmers with dyslexia do not usually like to draw attention to themselves.
- Conscientiousness: programmers with dyslexia are very conscientious.
- Aggressiveness: programmers with dyslexia are relatively serene people and are not aggressive.
- On the other hand, the major findings for programmers with dyslexia can be said to be related to certain physical behaviors that may or may not affect their programming performance. The most significant are related to:
 - Transposition of letters when reading or writing: this is the most common behavior and very patent when working with a text-based programming language.
 - Short-term memory impairment: this shows up as having to continuously reread recently written code.
 - Insomnia: adults with dyslexia usually have trouble sleeping, but this is not a common trait among the surveyed dyslexic programmers.
 - Visual stress: programmers with dyslexia have a serious problem with screens full of text and interfaces with some background and font colors, font sizes and character and line spacing as they easily get sidetracked and lose interest in the job they are doing.

2) *Programming-Related Variables*: Alsobhi and Abeyasinghe [7] claim that dyslexia is associated with a normal or above average intellectual coefficient and that people with this functional diversity are usually better software developers than regular people. However, it was found that there are

skills, behaviors and difficulties that may or may not affect their performance in programming-related tasks. The most significant are:

- Analytical skill: Alsobhi and Abeysinghe's findings were confirmed, especially with respect to the analytical skill.
- Problem identification: despite the above finding, 20.0% of programmers with dyslexia were found to have trouble performing this task.
- Command use and specific syntax handling: 20.3% of programmers with dyslexia stated that they find specific text-based programming language commands and syntax confusing to use.

B. Selection of Textual Programming Language

51 textual programming languages were identified as being commonly used by the 351 programmers surveyed. A first classification was made to determine the language to be included in subsequent studies. The variables Purpose and Frequency helped to make this classification:

- Purpose: To classify language in general or specific.
- Frequency: number of participants in the survey that have used the language.

Table I shows the use frequency and percentage of the 9 most used textual programming languages.

After classifying and analyzing 9 textual programming languages identified as the most used by 351 programmers, it was determined that Java would be the ideal language to be included in our model design.

C. Analysis of Visual Programming Languages

The selected 31 visual programming languages were provisionally classified according to the following variables to determine the languages to be included in later analyses:

- Purpose: whether the language has general or specific purpose.
- Validity: this variable was selected as there may be some special-purpose programming languages that may warrant inclusion in the final study. Possible values: Yes, No.

TABLE I. USE FREQUENCY OF TEXTUAL PROGRAMMING LANGUAGES

Textual Programming Languages	Frequency	Percentage
Java	293	83.48
C	229	65.24
JavaScript	161	45.87
C++	149	42.45
Python	134	38.18
PHP	114	32.48
C#	102	29.06
Visual .Net	80	22.79
M (MATLAB)	16	4.56

- Applicability: the language to be selected has to be currently applicable for use in practice. Possible values: Yes, No.

Table II shows the preliminary classification according to the above variables based on the information gathered about the 31 languages.

After this preliminary analysis, the classification was summarized as shown in Table III. We originally intended to consider only general-purpose languages for the subsequent study, as the aim was to preselect a language capable of performing complex programming tasks. However, since some special-purpose programming languages were found to be relevant, popular and have important features, the most significant ones were considered.

Although Blockly, Stencyl, Code and Subtext are tools highly relevant, they were not selected because they are not regarded as programming languages as such or run as a web page.

TABLE II. CLASSIFICATION OF VISUAL PROGRAMMING LANGUAGES

Visual Programming Language	Analyzed Variables		
	Purpose	Validity	Applicability
Alice [8]	Special	Yes	Yes
Appinventor [9]	General	Yes	Yes
Ark [10]	Special	No	No
Blockly [11]	Special	Yes	No
Code [12]	General	Yes	No
Cube [10]	Special	No	No
Drakon [13]	Special	No	No
Fabric [14]	Special	Yes	No
Forms/3 [10]	General	No	No
G [15]	Special	Yes	No
GameSalad [16]	Special	Yes	No
Max [17]	Special	Yes	No
Netlogo [15]	Special	Yes	No
Nxt-g [18]	Special	No	No
Kodu [19]	Special	Yes	Yes
Lava [20]	General	No	No
Limnor [21]	General	No	No
Prograph [10]	General	No	No
Pure Data [22]	Special	Yes	No
Quartz Composer [23]	Special	Yes	No
Robolab [24]	Special	Yes	No
Scratch [25]	Special	Yes	Yes
Simulink [12]	Special	Yes	No
Snap! [26]	General	Yes	Yes
Stencyl [27]	Special	Yes	No
Subtext [28]	General	Yes	No
Toontalk [29]	General	No	No
Vee [30]	Special	Yes	No
Vipr [10]	General	No	No
Vissim [31]	Special	Yes	No
V V V V [32]	Special	Yes	No

Note that the Drakon language is not considered applicable because it is related to a Russian project, and all the documentation that we have found is in Russian.

In total, we selected five VPLs: two were general-purpose, applicable and relevant languages (AppInventor and Snap!) and three were special-purpose, applicable and relevant languages (Alice, Kodu and Scratch).

Table III present a summary of the primary classification of visual programs languages.

After selecting the VPLs, we analyzed some of the technical and facilitative characteristics for programmers with dyslexia. They are described below.

1) Technical Characteristics: The following characteristics were selected for inclusion in the final classification because they are highly relevant to the future research to be developed:

- Implementation language: this characteristic refers to the language(s) in which the VPL was developed. Possible values: Java (J), Microsoft XNA (X), Kawa (K), Squek (S).
- Operating system: this characteristic refers to the operating systems on which the language runs. Possible values: Windows (W), MacOS (M), Linux (L), Android (A), Xbox 360 (X).
- Flow control: the effectiveness of a programming language largely depends on whether the runtime flow can be changed by means of control structures: close screen (C_S), close screen with result (C_SR), count (C), do (D_●), do in order (D_●), do together (D_T), each in together (E_IT), for (F), for each (F_E), for each in (F_EI), forever (F_E), for range (F_R), get start text (G_ST), if (I), if then else (I_TE), repeat (R), repeat until (R_U), when (W_H), while (W).
- License type: with a view to future project development decisions, it is important to determine whether the VPL has an open or closed source license. Possible values: open source license (●), closed source license (C).
- Visual building method: the building method refers to how the program is created. Possible values: simple drag and drop (DD) when there are several types of notchless blocks that the user drags and drops to create the code; puzzle drag and drop (PIDD), where most of the blocks have notches and have to fit perfectly like the pieces of a jigsaw; visual (V), when there is a set of icons governing specific program construction tasks.

TABLE III. SUMMARY OF THE VPL CLASSIFICATION

Classification	Number
Nonapplicable special-purpose VPLs	4
Irrelevant special-purpose VPLs	15
Applicable and relevant special-purpose VPLs	3
Nonapplicable general-purpose VPLs	5
Irrelevant general-purpose VPLs	2
Applicable and relevant general-purpose VPLs	2

- Type systems: data typing is closely related to the program runtime. There is a lot of debate whether static or dynamic type systems are more efficient [33]. Possible values: static (S), dynamic (D), static-dynamic (SD).
- Concurrent programming: this refers to programming notations and techniques to express potential parallelism and solve problems more efficiently [34]. Possible values: concurrent programming is enabled (Yes), concurrent programming is not enabled (No).
- Procedural abstraction: this refers to whether the language can decompose the program into modules, functions or subprograms. Possible values: procedural abstraction is enabled (Yes), procedural abstraction is not enabled (No).
- Data abstraction: this refers to whether the programming technique can be used to invent or define new data types (user-defined data types) adapted to the application to be implemented [35]. Possible values: data abstraction is enabled (Yes), data abstraction is not enabled (No).
- Other language: The visual programming language allows to generate an equivalent code in a text-based language or metalanguage, either directly or through external processes. Possible values: Java directly (J_D), XML directly (XML_D), XML by external conversion (XML_E), does not allow conversion (NP).

2) Facilitative characteristics of the visual programming language for programmers with dyslexia: Based on the profile of programmers with dyslexia, we identified some VPL characteristics that may help to improve the performance of programmers when working with any of these languages. They have been grouped into three categories which are described below.

a) Characteristics related to the transposition of letters when reading or writing: VPLs convey information more visually than text-based languages [36]. However, text is still a feature of many VPLs, and there are three key issues related to its use that warrant analysis:

- Nodes: it is important to analyze whether moving the mouse over or secondary clicking on onscreen nodes or elements delivers information. Possible values: detailed information on all nodes and elements (DT), detailed information on some nodes and elements (DA), basic information on all nodes and elements (BT), basic information on some nodes and elements (BA), a combination of basic and detailed information (CD), no information on nodes and elements (ND).
- Identifier names: this characteristic refers to how the VPL manages variable names, constants, types, subprograms. Possible values: user can define identifier names (Y), user cannot define identifier names (N).
- Expressions: this characteristic refers to aspects like the definition and use of mathematical formulae. Possible values: there are icons for creating

expressions (Y), there are no icons for creating expressions and they have to be input as text (N), the VPL has a mixed system for creating expressions (icons and text) (M).

- Language: people with dyslexia find it hard to interact with a language other than their native tongue. On this ground, language was established as a key characteristic for this analysis. This characteristic will specify the languages in which the language interface can be visualized: English (E), Spanish (S), another language (A), multi-language (dynamic change) (M).

b) Characteristics related to visual stress identified in the profile of computer programmers with dyslexia: Nonscalability is a factor to be very much taken into account when selecting a VPL, insofar as VPLs become unmanageable when describing very large programs [37]. Scalability issues are:

- Diagram complexity: this characteristic refers to how complex the diagrams generated by the language are. Possible values: high complexity (H), medium complexity (M), low complexity (L).
- Zoom capability: this characteristic is related to the capability of the environment to zoom in or out. Possible values are: any part of the environment can be zoomed (Y), zoom is not enabled (N), only some elements can be zoomed (P).
- Level of detail (L/D): this is a similar characteristic to zoom governed in this case by the concept of fisheye, that is, the magnification of a specific user-defined area of the language. Possible values: L/D is enabled (Y), L/D is not enabled (N).
- Representation viscosity: this refers to a representation's resistance to local changes. For example, as objects are added to a representation, it may be necessary to reorganize the entire diagram for the purposes of legibility. Possible values: no local changes are enabled (H), some changes can be made subject to constraints (M), there are no limits on changes (L).
- Representation density: this variable evaluates the number of pixels per node and the useful information per square centimeter that are used in all the onscreen representations. Possible values: there is a small number of pixels per node and a large amount of information per square centimeter (H), there is a moderate number of pixels per node and a moderate amount of information per square centimeter (M), there is many pixels per node and a very small amount of information per square centimeter or this information is displayed after an additional mouse action, resulting in rudimentary screen management (L).
- Icon dictionary: the language provides an icon gallery that can be used as building blocks for constructing objects. Possible values: there is an icon dictionary

(Y), there is no icon dictionary (N), there are icons for some tasks (S).

- Syntax: the language has a technique for reducing syntax errors. Possible values: block grouping is enabled (Y), no technique is provided (N).

c) Accessibility-related characteristics: Background and font colors, font sizes, character and line spacing have the potential to reduce the effects of the functional diversity of people with dyslexia [38]. These values are evaluated by the following characteristic:

- Adaptability configuration: this characteristic is evaluated if the VPL enables changes to the visual settings according to certain adaptability parameters. Possible values: all the above element settings can be changed (H), only font and background settings can be changed (M), only background settings can be changed (L), none of the above element settings can be changed (N).

D. Selected Visual Programming Languages

In the following we report the key features of the five visual programming languages selected for the final analysis.

1) Alice: This is a free educational tool to help create an animation, story, interactive game or video within a 3D programming environment. It was designed as a language for introducing a student to object-oriented programming and teaching the key programming concepts to create simple animated films and videogame.

This is a Carnegie Mellon University project sponsored by Oracle, Electronic Arts, Sun Microsystems, DARPA, Intel, Microsoft, NSF and NR [8].

2) App Inventor: This is a visual programming language to simplify the visual web-based creation of applications for the Android operating system. Users log in at <http://appinventor.mit.edu> to load the home page. This page basically provides access to several resources for educators, supplies usage information and tutorials, as well as building blocks and a set of basic tools that can be used to create mobile applications.

It was originally developed by Google, and, with the sponsorship of several companies, became a Web service administered by personnel from the MIT Centre for Mobile Learning in 2009. It is now called the MIT App Inventor community and has over three million users from 195 countries [9].

3) Kodu: This is a visual programming language designed to be accessible for children to create games for PC and Xbox in a 3D simulation environment. It is also used as a tool for teaching children to be creative.

Kodu can be used to teach creativity, problem solving, storytelling, as well as programming. Young children as well as adults with no design or programming skills can use Kodu to

make a game. It was created by Microsoft's FUSE Labs in 2006 and was originally known as Boku [19].

4) **Scratch**: Scratch primarily targets education through game creation, helping users to think creatively, reason systematically, work collaboratively and develop mental skills by learning from programming without necessarily knowing how to program. It has become very popular in young learner and adult education thanks to its computational thinking-related features.

Scratch enables users to use event-driven programming with multiple active objects called sprites. Sprites can be drawn as vector graphics or bitmaps from the Scratch website using a simple paint editor, which is part of the project, or can be imported from external sources. This is a product of the Lifelong Kindergarten Group at the MIT Media Laboratory and was officially released in 2005 [25].

5) **Snap!**: It provides a visual and clear interface for learning to program simple scripts from the browser. Each function in Snap! is defined as a jigsaw piece. It is ideal for anyone who does not know how to program or is a novice programmer. More advanced programmers regard it as a practical tool for making outlines and taking notes of ideas to program ahead of time.

Snap! is a language funded and promoted by the University of California, Berkeley [26].

IV. DISCUSSION

A. Analysis of the Characteristics of the Selected Visual Programming Languages

Based on the characteristics described in section III, we analyzed the five selected VPLs to determine the level of compliance of each language. The results obtained are summarized in the following sections:

1) **Technical characteristics**: Table IV shows each of the values analyzed in accordance with Section III.

2) **Facilitative characteristics of the visual programming language for programmers with dyslexia**: The characteristics that are analyzed under this point are specifically related to the key problems that programmers with dyslexia face with computer use. Depending on the reported values, these characteristics may lessen or heighten the identified problems. On this ground, they play a very important role in the analysis of the selected languages. They are illustrated in Table V.

After classifying the five most optimal VPLs, each language was briefly analyzed to determine the best language for model design.

It was found that Kodu is the least suited VPL for selection as the pilot for improving the performance of programmers with dyslexia, because, in the first place, its technical characteristics suggest that it is not a strong language for complex programming processes.

TABLE IV. TECHNICAL CHARACTERISTICS OF VISUAL PROGRAMMING LANGUAGES

Characteristics	Languages				
	L1	L2	L3	L4	L5
Implementation language	J	K, J	X	S	S
Operating system	W, L, M	W, L, M, A	W, X	W, M, L	W
Flow control	D_I, C, I_TE, W, D_T, E IT	I, FE, F_RW, G_ST, C_S, CSR	D, W_H	I_TE, R, R_U, F_E	I, R, F
License	●	●	●	●	●
Visual building method	DD	DDP	V	DDP	DD
Type system	S	D	SD	S	S
Concurrent programming	Yes	Yes	Yes	Yes	No
Procedural abstraction	Yes	Yes	No	No	Yes
Data abstraction	Yes	Yes	No	No	Yes
Text-based language	J_D	J_E	N_P	XML_E	XML_D

L1 Alice L2 AppInventor L3 Kodu
L4 Scratch L5 Snap!

On the other hand, it is missing some of the facilitative characteristics for programmers with dyslexia, such as syntax management, low representation viscosity, and nodes are over documented.

The second least suited language according to the analysis is Snap!, as, like Kodu, it is missing key characteristics for complex programming processes. With regarding the facilitative characteristics for programmers with dyslexia, however, it was found that this language has some good points that this population group might find helpful.

TABLE V. FACILITATIVE CHARACTERISTICS FOR PROGRAMMERS WITH DYSLEXIA

Characteristics	Languages				
	L1	L2	L3	L4	L5
Nodes	ND	ND	DT	ND	ND
Identifier names	Y	N	N	Y	Y
Expressions	M	N	N	Y	Y
Language	E, S	M	M	M	M
Diagram complexity	H	M	M	M	M
Zoom capability	N	N	Y	N	P
Level of detail	N	N	N	N	N
Representation viscosity	M	H	L	M	M
Representation density	H	L	L	H	H
Icon dictionary	N	N	S	N	N
Syntax	Y	Y	N	Y	Y

L1 Alice L2 AppInventor L3 Kodu
L4 Scratch L5 Snap!

Even though App inventor has sound technical characteristics for undertaking complex programming processes, it was rejected because of its dynamic type system and high representation viscosity. This is a very important variable for handling the visual stress to which programmers with dyslexia are prone.

The languages Alice and Scratch are equally powerful. This determines them as possible languages for inclusion in the model design. However, the fact that Scratch does not handle the abstraction of procedures or data puts it at a disadvantage with regard to Alice as a good language for complex programming tasks.

Alice would appear a priori to be the language that programmers with dyslexia are most likely to be comfortable working with in preference to a text-based programming language. Even though its diagrams are quite complex, it has visual characteristics that may be of assistance to programmers with dyslexia, and it has the option of minimizing syntax errors, which is one of the biggest problems that programmers with dyslexia have when developing software.

V. EXPERIMENT

An experiment was conducted to help determine the preferences, strengths and performance differences of a group of programmers with and without dyslexia. The participants used two different programming environments to develop basic programs.

A. Sample Design

For this experiment we used a convenience nonprobability sample of 44 people. They were selected from the participants of the first study that showed interest to collaborate in the research and that had basic knowledge of Alice.

B. Inclusion criteria

The following criteria were defined as requirements to be part of the experiment: Consent to participate and basic knowledge of the Alice programming language.

C. Exclusion criteria

Not passing a basic course in Alice was considered an exclusion criterion. 5 people did not pass the course, so the final sample had 39 individuals.

D. Experimental design

The experiment has been conducted to validate the use of Alice as a tool to improve the motivation levels of programmers with and without dyslexia when working as a team. The experiment was divided into 3 phases:

1) *Phase one. Dyslexia risk test.* We used “DyTECTive”, a dyslexia test created by Change Dyslexia [37]. 15 out of the 39 participants (38.5%) claimed to have symptoms of dyslexia. All participants used DyTECTive and 10 out of the 39 (25.6%) were identified to be at risk of having dyslexia. Of these 10 participants, 9 belonged to the group claiming dyslexia.

2) *Phase two. Program development.* Four exercises were defined. Two tests were applied at this stage. In each test the same exercise was developed (first in one language and then in the other). At the end of each exercise, the data corresponding to the variables presented in Table VI, were recorded.

a) *First test.* Participants developed a computer program in Alice and then in Java. Maximum time: 60 minutes per program.

b) *Second test.* Participants developed a computer program in Java and then in Alice. Maximum time: 120 minutes per program.

3) *Phase three.* Information-gathering instrument. The results obtained in the previous phase and other information that it shows in table VII, were recorded through an electronic survey.

E. Results

Given that the sample size is less than 50 people, a Shapiro-Wilk normality test was applied to analyze the quantitative data [39]. This data, reported for all quantitative variables, have a normal distribution, except for the Age and Number_Errors_2 variables which do not have a normal distribution.

1) *Age.* The age of the participants has a median of 23 (range from 17 to 53), ($p=0$). This is expected to happen given that most participants are university students.

2) *Sex.* 17.9% of the participants are female and 82.1% are male.

3) *Java knowledge.* The distribution is: low= 5, medium= 31 and high= 3.

4) *Alice knowledge.* The distribution is: low= 8, medium= 28 and high= 3. Both distributions are similar.

5) *Educational level.* The distribution is: 77.0% of the participants are students and 23.0% are professionals.

6) *Development Time.* Table VIII shows a summary of the number of programs finished and unfinished, classified by programmers with and without dyslexia risk.

TABLE VI. EFFICIENCY VARIABLES

Variable name	Definition	Possible Values
Development_Time_1 Development_Time_2 Development_Time_3 Development_Time_4	Determines the time in minutes taken by a subject to develop the experiment	Integer Value
Completed_1 Completed_2 Completed_3 Completed_4	Determines the completion of the requested program	Yes Not
Accuracy_1 Accuracy_2 Accuracy_3 Accuracy_4	Determines the level of accuracy of the obtained results according to the initial requirements	Percentage of test cases
Number_Errors_1 Number_Errors_2 Number_Errors_3 Number_Errors_4	Determines the number of faults detected in the generated code	Integer value

TABLE VII. GENERAL AND SATISFACTION VARIABLES

Variable name	Definition	Possible values
Java_level	Determines the level of knowledge in the use of the Java programming language	Low Medium High
Alice_level	Determines the level of knowledge in the use of the Alice programming language	Low Medium High
Dyslexia	Determines if a subject test positive for dyslexia	With risk No risk
Educational_level	Determines whether a subject is professional or not.	Student Professional
Anxiety_J	Determines the level of anxiety of a programmer in the use of the Java language.	Integer value
Anxiety_A	Determines the level of anxiety of a programmer in the use of the Alice language.	Integer value
Concentration_J	Determines the level of concentration in the use of the Java language.	Integer value
Concentration_A	Determines the level of concentration in the use of the Alice language.	Integer value
Syntax_J	Determines the level of syntax error in the use of the Java language	Integer value
Syntax_A	Determines the level of syntax error in the use of the Alice language	Integer value
Visual_Satisfaction	Determines the level of user satisfaction using a visual programming environment.	Integer value
Text_Satisfaction	Determines the level of user satisfaction using a textual programming environment.	Integer value

The measures of central tendency in relation to the time taken to solve the completed programs are listed below:

- Programmers at risk for dyslexia who completed the first exercise in Alice took a median of 36 minutes (CI= 32.5-41.5), while dyslexia risk-free programmers who completed the first exercise in Alice took a mean of 37 minutes (SD= 6.3).
- Programmers at risk for dyslexia who completed the first exercise in Java took a mean of 43.3 minutes (SD= 7.2), while dyslexia risk-free programmers who completed the first exercise in Java took a mean of 45.3 minutes (SD= 7.2).

TABLE VIII. SUMMARY OF PROGRAMS FINISHED AND UNFINISHED, CLASSIFIED BY PROGRAMMERS WITH AND WITHOUT DYSLLEXIA RISK

Exercise	With dyslexia risk		Without dyslexia risk	
	<i>Finished</i>	<i>Unfinished</i>	<i>Finished</i>	<i>Unfinished</i>
1 st Exercise (Alice)	10	0	24	5
1 st Exercise (Java)	9	1	24	5
2 nd Exercise (Java)	7	3	20	9
2 nd Exercise (Alice)	9	1	19	10

- Programmers at risk for dyslexia who completed the second exercise in Java took a mean of 71.6 minutes (SD= 20.4), while dyslexia risk-free programmers who completed the second exercise in Java took a mean of 81.1 minutes (SD= 25.1).
- Programmers at risk for dyslexia who completed the second exercise in Alice took a mean of 69.3 minutes (SD= 20.1), while dyslexia risk-free programmers who completed the second exercise in Alice took a mean of 85.2 minutes (SD= 24.5).

7) *Number of errors*. The number of errors presented in the development of the incomplete programs at the end of the allotted time were:

- Number_Errors_1 variable. In the development of the first programs in Alice, no programmer with dyslexia risk presented errors. On the other hand, five programmers without dyslexia risk presented a mean of 4 errors (SD= 2.4).
- Number_Errors_2 variable. In the first programs development in Java, one programmer with dyslexia risk presented 6 errors and 10 programmers without dyslexia risk presented a mean of 6 errors (SD= 3.5).
- Number_Errors_3 variable. In the second programs development in Java, three programmers with dyslexia risk presented a mean of 14 errors, (SD= 3.5) and 9 programmers without dyslexia risk presented a median of 4 errors (CI= 6-24).
- Number_Errors_4 variable. In the second programs development in Alice, one programmer with dyslexia risk presented 5 errors and 10 programmers without dyslexia risk presented a median of 3 errors (CI= 2-7).

8) *Accuracy*. Four test cases were performed per each finished exercise. With a Likert scale of 1 to 5, where 1 equals 0% accuracy, 2 - 25% (1 correct result), 3 - 50% (2 correct results), 4 - 75% (3 correct results) and 5 - 100% (4 correct results). Table IX shows the main findings.

9) *Anxiety, concentration and level of syntax error*. Given the importance of these characteristics, which were identified in the profile of programmers with dyslexia[6], they were questioned with a Likert scale scored from 1 to 5 (1 being understood as totally agreed and 5 as totally disagreed).

TABLE IX. SUMMARY OF ACCURACY LEVEL CLASSIFIED BY PROGRAMMERS WITH AND WITHOUT DYSLLEXIA RISK

Exercise	Programmers with dyslexia risk			Programmers without dyslexia risk		
	<i>Likert scale value</i>			<i>Likert scale value</i>		
	5	4	3	5	4	3
1 st Exercise (Alice)	100%			61.9%	33.3%	4.8%
1 st Exercise (Java)	77.8%	22.2%		62.5%	29.2%	8.3%
2 nd Exercise (Java)	57.1%	28.6%	14.3%	70%	25%	5%
2 nd Exercise (Alice)	77.8%	22.2%		68.4%	10.5%	21.1%

To establish the level of agreement, values 1 and 2 and values 4 and 5 were grouped together. The main findings are:

- To the affirmation: When I program in Java I often feel anxious, 80% of programmers at risk for dyslexia agreed and 6.9% of programmers at risk-free for dyslexia agreed.
- To the affirmation: When I program in Java I suffer from low concentration, 90% of programmers at risk for dyslexia agreed and 10.3% of programmers at risk-free for dyslexia agreed.
- To the affirmation: When I program in Java I make many syntax errors, 90% of programmers at risk for dyslexia agreed and 51.7% of programmers at risk-free for dyslexia agreed.
- To the affirmation: When I program in Alice I often feel anxious, 80% of programmers at risk for dyslexia disagreed, and 69% of programmers at risk-free for dyslexia disagreed.
- To the affirmation: When I program in Alice I suffer from low concentration, 100% of programmers at risk for dyslexia disagreed, and 75.9% of programmers at risk-free for dyslexia disagreed.
- To the affirmation: When I program in Alice I make many syntax errors, 100% of programmers at risk and at risk-free for dyslexia disagreed.

10) Visual and textual satisfaction. For these variables we used “The System Usability Scale” [38] which measures satisfaction on a scale from 0 to 100 based on 10 predefined questions. The main findings are:

- Alice satisfaction. This present a general mean of 68.7 (SD 6.7), a mean of 71.3 (SD 4.5) to programmers with dyslexia risk and a mean of 67.9 (SD 7.1) to programmers without dyslexia risk.
- Java satisfaction. This present a general mean of 69.7 (SD 8.4), a mean of 64.3 (SD 3.9) to programmers with dyslexia risk and mean of 71.6 (SD 8.7) to programmers without dyslexia risk.

F. Analysis of results.

Regarding the number of completed exercises within the defined time, it can be established that programmers with a risk of dyslexia show a higher level of success than the programmers without risk of dyslexia. In addition, the time taken to solve the exercises, according to the central tendency measures, is less in the programmers with risk of dyslexia, regardless of the language they use.

When analyzing the behavior of programmers who did not complete the exercises within the allotted time, it is observed that the highest frequency of unsuccessful was presented in the programmers without dyslexia risk, independent of the analyzed programming language.

The programmers with risk of dyslexia are more anxious programming in Java than the programmers without risk of dyslexia, while the programmers with risk of dyslexia are less anxious programming in Alice than the other programmers.

Java programmers with risk of dyslexia have low levels of concentration while programmers without risk of dyslexia have medium levels of concentration. Alice programmers with risk of dyslexia have a high degree of concentration while programmers without risk of dyslexia have a medium-high level of concentration.

Java programmers with risk of dyslexia made more syntax errors than programmers without risk, while Alice programmers do not report frequent syntax errors.

Overall, there is more satisfaction with the use of Java than with Alice, but programmers with risk of dyslexia are more satisfied with the use of Alice than with Java, while programmers without risk for dyslexia are more satisfied with the use of Java than with Alice.

VI. CONCLUSIONS

The results of our study allow us to draw some conclusions that are described below:

Although there are no studies that show that people with dyslexia tend to have a higher mental coefficient, during the development of the different studies it has been observed that computer programmers with dyslexia show better analytical skills when it comes to solving a certain problem, presenting higher levels of precision in the required results.

Programmers with symptoms of dyslexia when developing software in a textual programming language can often be affected by aspects of their personality such as anxiety and low concentration when performing a task. There is also some stress that come from some aspects of textual languages. This significantly affects their work and their personal performance.

However, the results of the experiment reinforce the theories proposed in the profile of the programmers with dyslexia. On the other hand, the high percentages presented in the results of the experiment support the idea that computer programmers with dyslexia perform better and have better skills in the development and monitoring of computer programs in a visual environment and support the Dixon hypothesis presented in section III. This allows us to state that computer programmers with dyslexia perform better in a visual programming language than in a textual language.

As future work, we plan to conduct an experiment with a prototype implementing the designed model. The experiment will involve the participation of programmers with or without risk of dyslexia and the goal of this will be to measure the level of acceptance, the usability of the programming tasks using the prototype and the performance of the programmers when working in teams.

ACKNOWLEDGMENT

We would like to thank the Universidad Politécnica de Madrid (Spain), the Politécnico Colombiano Jaime Isaza Cadavid of the city of Medellín (Colombia) and all the people who have collaborated in the studies conducted to gather the information necessary to contextualize the research.

References

- [1] Rello, L. and Baeza-Yates, R. The presence of English and Spanish dyslexia in the Web. *New Review of Hypermedia and Multimedia*, 18, 3 (09/01; 2015/09 2012), 131-158. DOI=10.1080/13614568.2012.721401.
- [2] Dixon, M. Comparative Study of Disabled vs. Non-disabled Evaluators in User-Testing: Dyslexia and First Year Students Learning Computer Programming. In Stephanidis, C. ed. *Springer Berlin Heidelberg*, , 2007, 647-656.
- [3] Fuertes, J., González, L. and Martínez Loïc. Characterization of Programmers with Dyslexia. In Springer ed. *Computers Helping People with Special Needs*. Editorial Springer, Linz, Austria, 2016.
- [4] González, L. Encuesta tesis doctoral "Transformación bidireccional de código de software visual y texto, para mitigar los efectos de la dislexia en el desarrollo de software. <http://oa.upm.es/39155/>. (2016).
- [5] Waldo SL, McIntosh DE, Koller JR. Personality profiles of adults with verbal and nonverbal learning disabilities. *Journal of Psycho-educational Assessment* 17, (1999), 196-206.
- [6] Fuertes, J., González, L. and Martínez Loïc. Características de los programadores disléxicos. *Actas de Ingeniería*, 2 (2016).
- [7] Alsobhi, A. Y. and Abeysinghe, G. An evaluation of accessibility of e-learning for dyslexic students. 2013 International Conference on Current Trends in Information Technology (CTIT), (2013 2013), 1; 1-4; 4. DOI=10.1109/CTIT.2013.6749468.
- [8] Carnegie Mellon University. ALICE. An Educational Software that teaches students computer programming in a 3D environment. 2017, Febrero 17 (2017).
- [9] MIT Media Lab. MIT App Inventor. 2017, 02/17 (2015).
- [10] Boshernitsan, M. and Downes, M. Visual Programming Languages. In Anonymous *Visual Programming Languages: A Survey*. Computer Science Division (EECS, Berkeley, California, 2004, 6-23.
- [11] Google Inc. Google for education: A library for building visual programming editors. 2017, 01/19 (06/23/2016 2016).
- [12] MathWorks. Simulink. Simulación y diseño basado en modelos. 2017, 01/16 (2017), <https://es.mathworks.com/products/simulink.html>.
- [13] Mitkin, S., Ilyin, A., Zeeman, M., Dyadov, V. and Bachiashvili, V. Drakon. 2017, 02/07 (03/10/2006 2016), <http://drakon-editor.sourceforge.net/>.
- [14] Myers, A., Arden, ●., Liu, J. and Magrino, T. Fabric: Strong security, strong consistency, open and decentralized. 2017, 02/13 (06/30/2016 2016), www.cs.cornell.edu/projects/fabric/.
- [15] Wilensky, U. NetLogo. 2017, 01/23 (1999), <http://ccl.northwestern.edu/netlogo/docs/>.
- [16] GameSalad Inc. GameSalad teaches Computer Science by enabling students to easily and rapidly build great games. 2017, 01/2' (2014), <https://www.gamesalad.com/>.
- [17] Cycling '74. Max connects objects with virtual patch cords to create interactive sounds, graphics, and custom effects. 2017, 01/23 (2017).
- [18] Lego Mindstorms México. El lenguaje de programación NXT G 2. 2017, 01/21 (05/26 2016), <https://nxtmexico.wordpress.com/>.
- [19] Fuse Labs, M. R. Kodu Game Lab. 2017, 01/20 (04/20 2017), <https://www.kodugamelab.com/>.
- [20] Helpes. Lava (Lenguaje de programación). 2017, 01/20 (2016), [http://www23.helpes.eu/01103997/Lava\(lenguajeDeProgramacion\)](http://www23.helpes.eu/01103997/Lava(lenguajeDeProgramacion)).
- [21] Limnor Studio. Limnor Studio for visual programming. 2017, 01/23 (04/16 2017), <http://www.limnor.com/LimnorForum.html>.
- [22] PD Community. Pure Data. 2017, 01/23 (10/01/2015 2015).
- [23] Apple Inc. Introduction to Quartz Composer Custom Patch Programming Guide. 2017, 01/23 (03/24/2010 2010).
- [24] LEGO Engineering. Robolab. 2017, 01/23 (2017), <http://www.legoengineering.com/category/robolab/>.
- [25] MIT Scratch Team. About Scratch. 2017, 01/23 (2017), <https://scratch.mit.edu/about>.

- [26] Brian, H. and Möning, J. Snap! Build Your Own Blocks. , 01/23/2017 (2017), <https://snap.berkeley.edu/SnapManual.pdf>.
- [27] Stencyl, L. Stencyl. 2017, 01/23 (2017), <http://www.stencyl.com/>.
- [28] Jonathan, E. Subtext: uncovering the simplicity of programming. 2017, 01/23 (2017), www.subtext-lang.org/.
- [29] Ken, K. ToonTalk - Making programming child's play. 2017, 01/23 (2009).
- [30] Agilent Technologies. Agilent VEE Pro. 2017, 01/23 (2017), <http://es.freownloadmanager.org/Windows-PC/Agilent-VEE-Pro.html>.
- [31] SolidThinking. VisSim. 2017, 01/23 (2017), <http://web.solidthinking.com/vissim-is-now-solidthinking-embed>.
- [32] Joreg. V V V V. 2017, 01/23 (07/17 2014), <https://vvvv.org/>.
- [33] J. K. Ousterhout. Scripting: higher level programming for the 21st Century. Computer, 31, 3 (1998), 23-30. DOI=10.1109/2.660187.
- [34] Schneider, F. B. *On Concurrent Programming*. Springer New York, , 2012.
- [35] Black, P. E. Abstract data type. Dictionary of Algorithms and Data Structures [online], 2007, 04/01 (12 de Diciembre de 2016 2005).
- [36] Canfield Smith, D. Pygmalion, A Creative Programming Environment. Birkhauser-Verlag, Basel, and Also Stanford University AI Memo 260, , CS Dept. Stan-CS-75-499, (1977 and 1975).
- [37] W. Citrin, R. Hall, C. Santiago and B. Zorn. Addressing the scalability problem in visual programming through containment, zooming and fisheyeing. In Anonymous *1998 IEEE Aerospace Conference Proceedings (Cat. No. 98TH8339)*. 0. , 1998, 189-202 vol.4.
- [38] Rello, L., Kanvinde, G. and Baeza-Yates, R. Layout Guidelines for Web Text and a Web Service to Improve Accessibility for Dyslexics. In Anonymous *Proceedings of the International Cross-Disciplinary Conference on Web Accessibility*. (Lyon, France,). ACM, New York, NY, USA, 2012, 36:1-36:9.
- [39] Shapiro, S. S. and Wilk, M. B. An analysis of variance test for normality (complete samples). Biometrika, 52, 3-4 (1965), 591-611. DOI=10.1093/biomet/52.3-4.591.