VFAST Transactions on Software Engineering
http://vfast.org/journals/index.php/VTSE@ 2022, ISSN(e): 2309-3978, ISSN(p): 2411-6246
Volume 10, Number 2, April-June 2022 *pp: 47-60*

A Comprehensive Survey and Analysis of Diverse Visual Programming Languages

Muhammad Idrees*[1], and Faisal Aslam[2]

[1]Department of Data Science, University of the Punjab, Lahore, Pakistan
[2]Department of Computer Science, University of the Punjab, Lahore, Pakistan.
*Corresponding author email: idrees@pucit.edu.pk

ABSTRACT

Visual Programming Languages (VPLs) provide ease of programming by reducing the need of manually typing code for programming. Although the existence of VPLs is almost as old as textual programming languages but they have not become a mainstream technology for developing professional programs. However, the recent introduction of web-based VPLs, such as Scratch and Snap, has reinvigorated the usefulness of VPLs. Today, there exist dozens of VPLs having diverse characteristics. However, a comprehensive analysis of these diverse visual programming languages has never been conducted. Such an analysis is required for identifying the strengths and weaknesses of VPLs, as well as to choose the most suitable VPS for the task in hand. To that end, this study has performed a comprehensive search of a large number of 40 VPLs and analyzed and compared these VPLS based on 14 characteristics.

KEYWORDS

Visual Programming Language, Survey, Novice Programmers, Characteristics of VPL

JOURNAL INFO

HISTORY: Received: March 10, 2022 Accepted: May 8, 2022 Published: May 15, 2022

INTRODUCTION

Early computing was limited to a few skilled users because a user had to memorize complicated operating system commands and understand the underlying hardware, even for performing trivial tasks. To make computing easier for the general public, Graphical User Interfaces (GUIs) were introduced as abstractions to the underlying hardware and operating system commands. A GUI allows non-technical computer users to perform otherwise complicated tasks by using simple mouse clicks and drags-and-drops, making computing accessible to the masses.

The introduction of GUIs has made basic computing tasks (such as word processing) more accessible to the masses. However, it has not yet mitigated the cumbersome coding effort required to develop new programs. To address this problem, Visual Programming Languages (VPLs) were introduced. A VPL refers to the category of programming languages where the program logic is represented using visual elements. A key benefit of using VPLs is that programmers can focus on devising the logic of a problem's solution instead of worrying about the syntax of the programming language and the low-level intricacies of the development platform. VPLs thus have the potential to significantly enhance a programmer's productivity by making programming easier. However, the programmer loses some control over the optimization of code, since the code generated by using drag-and-drop of visual elements might not be as optimized as the textual code written by a skilled programmer.

Over the years, numerous VPLs have been developed, each having its own strengths and weaknesses. Several studies have established that the choice of a programming language has a strong correlation with the productivity of a programmer [1], [2]. Therefore, the selection of a programming language plays a critical role in the success of a project. We argue that a comprehensive comparison of VPLs is desired in order to identify the most appropriate language for a given task. Idrees et al. [3] have described the evolution of VPLs in three generations and defined relationships between VPLs during their evolution.

In this paper, the detailed survey of VPLs mentioned in [3] and their key characteristics gathered from the first-hand data are presented. However, the list can be extended, and that study encouraged to extend the set of characteristics of their choice in order to compare and assess the appropriateness of VPLs. The key contributions of this study are the following:

- We have provided an overview of the existing attempts to study and compare VPLs.
- A comprehensive list of characteristics for VPLs that are identified during an extensive literature survey.
- For each VPL, every characteristic is assigned a numeric score on a scale of 0 to 1, where 0 represents the absence of the characteristic and 1 represents the presence of the characteristic in that VPL. A value between 0 and 1 indicates the partial presence of the characteristic.

Organization

The rest of the paper is organized as follows:

Section II presents related work published as research papers, student reports, and dissertations. Section III describes published work related to this paper and the criteria that we have used for the inclusion of a VPL. Section IV describes VPLs included in this paper while highlighting their important characteristics. It also discusses the characteristics used as the comparison criteria for the VPLs and assigns, along with justification, a score to every characteristic of each VPL. Finally, the limitations of the VPL ranking procedure, directions for future research and the drawn are presented in Section VI.

RELATED WORK

In 1986, Brad A. Myers [4] conducted the first survey paper on visual programming languages. It divides visual programming languages into three categories: a fully visual programming language, programming by example, and program visualization. The program visualization and programming by example both do not fit with our definition of visual programming languages given in Section 3. It is so because program visualization tools create a visual representation of a textual code, but do not allow writing new code using visual interfaces. The tools that support programming by example are used when given an intended output of a textual program, the corresponding textual program is automatically created. Although program visualization and programming by example are both useful techniques to learn programming, they are not the subjects of this paper. This survey also discusses some of the earliest visual programming languages such as PIGS, Piet, Prograph, Autoprogrammer, Pygmalion, SmallStar, and Rehearsal World. We could not find information (such as binaries, source-code or user-manual) for any of these visual programming languages except Prograph, which is now known as Marten [5] and is included in our survey.

In 1989, Brad Myers [6] wrote another historical survey of visual programming languages. This survey groups visual programming languages based on four characteristics: VPLs with a compiler, VPLs with an interpreter, VPLs that provide example-based programming, and domains of VPLs. Using these four characteristics, it lists more than 30 visual programming languages. However, we have no way of verifying today if all of these languages were visual according to our definition given in Section 3. It is so because most of these languages are obsolete now, and their code or documentation cannot be found. For instance, Fabrik [7] was developed in the 1980s, and by 1987 it was discontinued [8]. The only two languages from this survey that are still available are Prograph and LabView. However, Prograph is now available under the new name Marten [5].

Another survey paper of that era, written in 1992, talks about flowchart based programming and its benefits [9]. It provides an overview of more than a dozen early visual programming languages. However, unfortunately, as with other surveys of the 20th century, these VPLs did not survive the tests of time, and their binary or source code cannot be found now.

A visual programming survey written in 2004 [10] provides a lot of insight into the history of visual programming. According to this survey, the first visual programming language that allowed writing code using data-flow diagrams was developed in 1965 [11]. In 1975 pioneering work on iconic programming was carried out. The paper also talks about relatively recent visual programming languages, including VIPR, Prograph, PICT, Cube, and Forms/3. Unfortunately, all these languages were developed in the 1990s and are dead because neither their code nor their binaries are available now.

A useful literature survey of visual programming languages can be found in the Master thesis of Ruchika Singh, written in 2008 [12]. Although the thesis is about a visual tool called MosGen, but it also provides a reasonable comparison of some VPLs, including BACCII, FLINT, SIVIL, RAPTOR, B#, VPF, CMCVLS, VAP, HomeCI, and Progranimate. The binaries of some of these languages are available today, and these have been included in our survey. We have not included VAP because it requires writing high-level code manually, which can be subsequently converted into typical textual language code and, thus, does not fulfil our inclusion criteria of Section 3. However, BACCII, FLINT, SIVIL, RAPTOR, B#, and Progranimate are included in our survey.

The PhD dissertation of Andrew S. Scott [13] that introduces Progranimate as a new VPL includes a survey of existing VPLs such as Flowchart Interpreter, Raptor, SFC editor, Visual Logic, The Iconic Programmer, Dev Flowcharter, B#, ProGuide, and Code Visual to Flowchart. Many of these VPLs that fulfil our inclusion criteria have been made part of our survey. The thesis also describes to the extent to which a VPL is visual, supports debugging visual code, has threads, etc.

Stelios Xinogalos [14] provides insight into 11 flowchart-based visual programming languages, including BACCII/BACCII++, FLINT, SFC Editor, RAPTOR, SICAS, SICAS-COL, H-SICAS, ProGuide, B#, Iconic Programmer, and Progranimate. The discussion on these VPLs in this 2013 paper includes whether a language is object-oriented or procedural if it provides animation of the logic, whether its binaries are available, and to which textual languages its code can be translated. Although this paper is closely related to our research, however, only four of the 11 languages discussed in the paper have their binaries available presently. Our survey is more comprehensive and provides a superset of information provided in this paper. Further, our paper discusses dozens of VPLs whose binaries are currently available and which can be used by programmers.

Hooshyar, Ma´ıen, and Masih [15] provide a brief survey of eight VPLs, including BACCII, Raptor, SFC Editor, DevFlowcharter, B#, CVF, and Progranimate. In fact, CVF is not a VPL as it is only capable of visually presenting a textual code as a visual flow chart. We have included the rest of the VPLs discussed in this survey in our paper while conducting a more thorough comparison.

In conclusion, the existing work has one or more of the following shortcomings: a) *Dated surveys*: The last dedicated survey was published several years ago, in 2004 [10]. Since then, all the studies primarily focus on proposing a VPL artifact while providing a brief overview of a few of the existing VPLs. b) *Inclusion-exclusion criteria*: The existing studies do not describe the criteria for shortlisting, including, and excluding the VPLs. c) *Comparison criteria*: The existing surveys do not rely on comprehensive criteria to compare the VPLs. d) Limited scope: A majority of existing studies discuss fewer than a dozen VPLs. In contrast, this is the largest ever and most comprehensive survey of several dozens VPLs.

VPL BACKGROUND

This section presents the background of the study by providing the definition of a VPL, the criteria used for the selection of VPLs and the VPLs that are included in this study. Note, that we introduced the definition of the VPLs in a previous study [3].

> ***Definition 1 (VPL):*** *A programming language is called VPL if it fulfils two conditions: it provides fundamental programming constructs (especially conditional constructs and loops), and a program logic can be implemented using drag-and-drop of visual elements.*

Criteria of VPL Selection

This study has included most of the VPLs that we found during our search, as mentioned in Table 1, which spans over two years. However, there is a possibility that a few VPLs may have evaded our exhaustive search. Our screening criteria to include the VPLs for which sufficient information is available related to their various characteristics. These criteria include the following:

- The VPLs that are missing basic programming constructs are excluded from the study.
- The VPLs that are not completely visual are also excluded.
- Nonetheless, the VPLs with a scarcity of information that is inadequate to assess the values of the characteristics are excluded.

For the inclusion of VPLs in our survey, this study has employed an established procedure that is widely used for conducting a systematic literature review [16]. It consists of three steps that are discussed below:

- Generating a list of candidate VPLs as mentioned in Table 1 results in a list of over 150 candidate VPLs.
- Screening the candidate list through the documentation of each candidate VPL.
- Screening the quality of the candidate list through cross-validation by a group of five associated researchers. This step is significant to compute the accurate score of the characteristics of the selected VPLs.

The 40 selected VPLs as per the above criteria are presented in Tables 3, 4 and 5. These VPLs are used in the rest of the study for analysis and comparison.

Table 1: Search terms and space.

Keywords	Visual programming languages, visual program, graphical programming languages, graphical programming, programming without typing, programming without keyboard, programing on handheld devices, kids programming, and programming tools for beginners.
Search space	ACM DL, IEEE Xplore, ScienceDirect, Springerlink, Google, Google-scholar, Wikipedia.

OVERVIEW OF VPLS

The binaries or related publications for most of the 40 VPLs are currently spread across a number of sources. Therefore, a fundamental challenge for the research community is to study and compare VPLs is to develop a unified source of the relevant literature and then synthesize it for the analysis and comparison. To that end, this section provides an overview of the 40 VPLs that we have collected from a comprehensive search. A brief description of key VPLs is as follows.

Flowgorithm [17], [18] is a general-purpose VPL that can be used to create flowchart representations of computing algorithms. It can translate a visual flowchart into eleven different textual programming languages, including C++ and Java. Flowgorithm is language independent but is not platform-independent; it is available only for Microsoft Windows operating system. Although it is not open source but is free for use. Figure 1 shows a simple program in Flowgorithm that identifies even and odd integers in the first 100 integers starting with 1, with its C++ equivalent code automatically generated by Flowgorithm's development environment.

Larp [19], [20] facilitates novice programmers to design program logic using flowcharts. Furthermore, a distinct feature of Larp is that it allows the representation of program logic as pseudocode. It is language independent but is limited to Microsoft Windows platforms only. Larp cannot generate code in any textual language; instead, it executes a visual flowchart directly. It is freely available but is not open source. Thus, it provides no mechanism to extend its functionality.

Raptor [21]–[23] was developed by the US Air Force to visualize computing algorithms. It is a language-independent VPL and is available for Microsoft Windows and Linux platforms. Raptor has the ability to generate textual code in four different programming languages: C, Java, Ada, C# and VBA. However, the generated textual code is not complete. Object Orientation and UML Designer

integration makes Raptor distinctive as compared to other VPLs. Its source is freely available under the US Air Force policy.

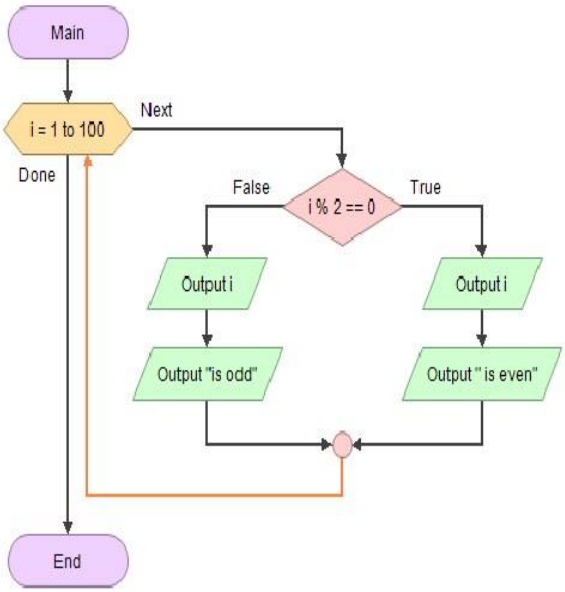

Figure 1: A Flowgorithm's program to identifying even and odd integers in the first 100 integers starting with 1

Table 2: Availability of VPL binary, year of first and most recent versions.

Sr. No.	VPL	Sr. No.	VPL	Sr. No.	VPL
1	Flowgorithm	2	Programinate	3	Flowcharts Interpreter
4	Raptor	5	Larp	6	Visual Logic
7	DRAKON	8	devFlowcharter	9	Iconic Programmer
10	B#	11	Marten (Prograph)	12	App Inventor for Android
13	FLINT	14	BACCII/BACCII++	15	The SFC Editor
16	SIVIL	17	Scratch	18	Snap
19	Tynker	20	Stencyl	21	Agent Sheets
22	CODE	23	EToys	24	GameSalad
25	Blockly	26	Touch Develop	27	BlockPy
28	VIPLE	29	Microsoft VPL	30	Lego Mindstorms Software
31	MBlock	32	Open Roberta	33	Pencil Code
34	Alice	34	StarLogo TNG	36	Beetle Blocks
37	Kudo	38	Dynamo	39	LabVIEW
40	Analytica				

Programinate [24]–[26] is a web-based general-purpose VPL implemented in Java. Like Flowgorithm, it can also be used to create flowchart representations of computing algorithms. It can translate visual code into six different textual programming languages, including Java, Visual Basic, and Pascal. Programinate is language and platform-independent. A useful feature of Programinate is that it executes a flowchart and the corresponding code synchronously, leading to a better understanding of the code behavior. Programinate is not open source but is free for use.

Flowcharter Interpreter (FI) [27] is a VPL that is based on a subset of C++. It can create flowchart representations of C++ programs; hence it is not language independent. It is also not platform-independent and is available for Microsoft Windows operating system only. FI is a free and open-source VPL.

Visual logic [19], [28] is a commercial tool that is not free. It provides minimal syntax to novice programmers to learn computer programming. It is language independent and generates code in Visual Basic and Pascal. Visual logic is available only for the Microsoft Windows platform as non-extensible software.

Marten (formerly known as Prograph) [5] is currently available only on Mac OS X. It is not a typical flowchart-based VPL and may be used to develop large programs. New methods in Marten are drawn on a canvas called cases. Initially, each case consists of two operations to receive inputs from other cases and to propagate data to other cases. More operations can be added and linked in between the two initial operations to develop program logic leading to a graphical structure. Marten is an object-oriented VPL and is currently free to download and use.

DRAKON [29] is a VPL from Russia. It was initially designed to specify requirements for software used to control a spacecraft by facilitating the creation of class diagrams and sequence diagrams. However, it can also be used to graphically represent the logic of computing algorithms. It is available as a family of languages like DRAKON-C, DRAKON-Java, etc. DRAKON building blocks are icons and macroicons. An icon is like an alphabet and a macroicon is a word based on icons. There are 27 icons and 21 macroicons in DRAKON. It generates code in several textual programming languages, including C, Java, Go and Python. It is free and open-source. Figure 2 shows a simple DRAKON program.

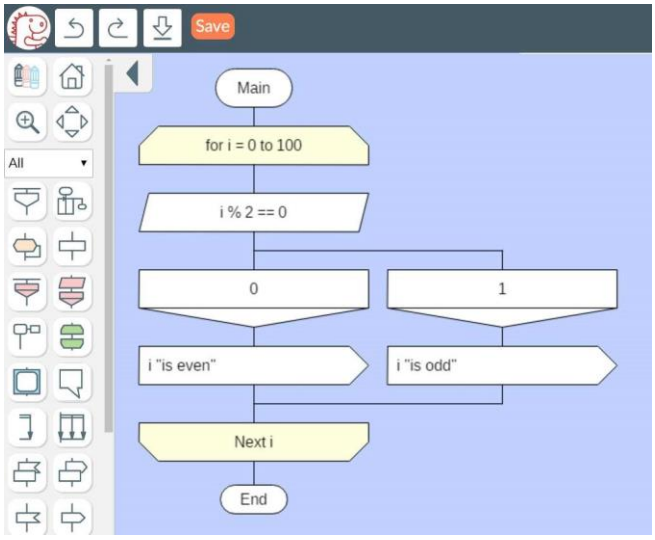

Figure 2: A simple Drakon's visual program

Iconic programmer [30] is a software tool that facilitates a novice programmer to quickly write a program by creating a flowchart using a drag-and-drop of icons. A drawback of Iconic Programming is that it does not provide user-defined types, functions, and even arrays. Iconic Programming is free to download and use on the Microsoft Windows platform, but it is not open source.

devflowcharter [31] is a Microsoft Windows-based free and open-source educational tool that can be used to develop a visual program in the form of a flowchart. It supports global variables, global constants, and functions. Furthermore, one may use the libraries of an underlined textual language while programming visually. It allows the declaration and use of user-defined data types. It also supports arrays. The main shortcoming of devflowcharter is that it requires the selection of a textual language such as Pascal, ANSI C, or TIBasic before creating a visual flowchart for the corresponding program in that language. Thus, it can produce language-dependent flowchart diagrams.

App Inventor for Android [32] was initially developed by Google but is currently maintained by MIT. It can be used to create simple apps for Android platforms using a drag-and-drop interface. Its visual interface provides a set of tools for beginners and intermediate programmers, as well as for educators. It provides a web-based interface for the development of apps and is, therefore, platform-independent. App Inventor can access most of the databases of a phone's functionalities and can invoke other apps. However, it cannot access the file system, has limited access to the web, and has limited user interface capabilities. Furthermore, it lacks polymorphic components, as functions developed for one app cannot be reused by other apps. It is a free and open-source tool available under the MIT license.

Touch Develop [33]–[35] by Microsoft is a free and open-source block-based VPL that makes computer programming available on touch devices, such as smartphones and tablets. It can run on various platforms, including web, iOS, and Android. It is primarily developed to visually code apps and games, but it also can be used to program general-purpose algorithms like searching and sorting. Apps developed in Touch Develop may be exported to several platforms, including Apache Cordova, Azure web, Office mix, and Node.JS [36].

B# [37], [38] is a VPL tool based on iconic programming for novice Pascal language programmers. It provides a drag-and-drop interface to create flowcharts and shows textual code as a flowchart that is being created. Hence, providing a real-time comparison between them. It supports basic programming constructs like assignment, loops, and conditions as well as helps in understanding syntax and errors. It facilitates the step-by-step execution of a program by highlighting the current instruction in execution and showing the changing states of variables. It is available for Microsoft Windows only. It is free and open-source.

BACCII [39] is a Microsoft Windows-based visual programming tool developed to teach computer programming to novice programmers. In BACCII, programs are built by drawing icons in a flowchart. These icons represent different programming constructs like a variable, loop, if/else, and input/output. After completing a flowchart, a user can generate syntactically correct code in high-level languages, including BASIC, C, and FORTAN [39], [40]. Later, object-oriented constructs were integrated into the system, and the extended language was named BACCII++ [40]. It generates syntactically correct code in C++. It provides an easy-to-use environment to novice programmers by hiding the details of language syntax and resulting errors. Although BACCII has historical significance but currently its

source and binary are unavailable.

Scratch [41]–[44] has been designed and developed at MIT primarily as a web-based VPL that also provides an offline editor to create games, animations, and stories. Scratch provides media-rich and network-based activities with addon capability of programming. The primary shortcoming of Scratch is that it cannot be used to develop general-purpose and complex software. It supports threads for parallelism but does not allow the use of procedures and functions. It is open source under GPLv2 and is available to users free of cost.

Snap [45], [46], a drag-and-drop browser-based VPL is based on Scratch and was developed at UC Berkeley. Unlike Scratch, which provides a set of building blocks, Snap facilitates its programmers to create their own blocks that can be used to build first-class procedures, first-class lists, etc. This added functionality makes Snap usable for writing complex programs as compared to Scratch. Snap is open source and is available free of cost.

Tynker [47], [48], a programming environment similar to those of Scratch and Snap, is aimed at teaching programming to children. It can be used to build games, control robots and drones, and explore STEM (Science, Technology, Engineering, Mathematics). Tynker website facilitates educators, students, and parents to teach coding to kids of ages 7 to 14. Tynker is a commercial project. It is neither open source nor free.

FLINT, aka Flowchart Interpreter (FLINT) [49] is a limited functionality tool developed in the late 1990's to teach programming concepts to novice programmers by eliminating the need to memorize syntactical details [49]. It uses an iconic interface for developing flowcharts, which can be interpreted and even debugged in the FLINT environment. It is a historical VPL whose source code and binaries are not currently available.

SIVIL [49], [50], like FLINT, is also a visual programming tool used to teach programming concepts to novice programmers. It too provides the compilation and debugging facilities but uses non-conventional icons, which makes it completely different from commonly used programming languages. Like FLINT, SIVIL is also a historical VPL whose source code and binaries are currently not available.

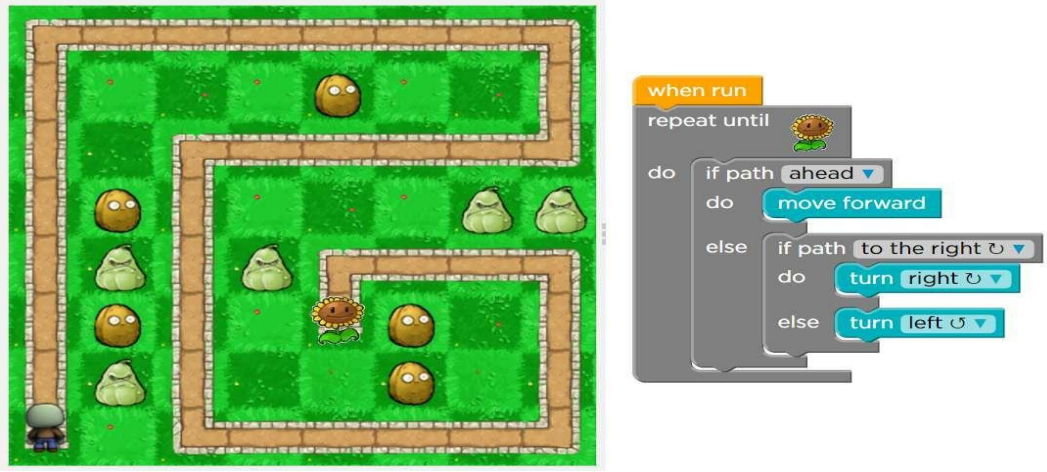

Figure 3: A visual program in code that moves an animated zombie

CODE [51] was designed and maintained by a non-profit organization supported by Microsoft, Google, Facebook, and others. It is widely used to teach programming in schools across USA [52]. It familiarizes students with the basic concepts of programming such as conditional constructs and loops using a graphical environment. It provides well-known gaming characters from popular games, including angry birds, plant vs. zombies, and Minecraft, to make coding fun for kids. Code is limited to teaching introductory programming and cannot be used for writing generic programs. The logic developed using the visual interface may be viewed in the JavaScript language syntax. It is open-source and free to use. Figure 3 shows a simple program in code that moves an animated zombie along a specific path.

The **SFC Editor** [53] is a VPL used to create structured flowcharts and produce their equivalent pseudocodes. It facilitates GUI based building blocks and dialogue windows for the basic structured flowcharting constructs such as sequence, selections, and iterations. It supports functions and subroutines. It was developed for the Microsoft Windows platforms only. We were unable to download and use it code-free or otherwise, as the download link was not accessible [54].

Stencyl [55]–[57] is primarily used to design 2D games like Alice using visual building blocks. These visual blocks are extensions of visual blocks available in Scratch as well as those available in Snap and Tynker. Stencyl is used to

teach programming to kids and novice programmers. It is neither open source nor free.

The development of **Etoys** [58] was influenced by Logo and Squeak. It is a child-friendly, media-rich, and object-oriented language that can be used to create graphics, animations, and sounds. It is extensively distributed on cheap laptops under the ”One Laptop Per Child” project [59]. It is platform-independent and can thus be used to program on almost any machine. Furthermore, it is available free of cost under the MIT open source license.

AgentSheets [60], [61] is a tool that provides domain-oriented visual programming environments and languages [62], [63]. It may be used to teach programming and related IT technologies to middle and high school students through game design. Its drag-and-drop environment enables students to quickly learn programming by developing simple games. AgentSheets can also be used to develop advanced games employing artificial intelligence. Its visual programs can be translated to Java. The AgentSheets environment is a computational grid where each cell may contain an agent consisting of a picture, an animation, a sound, or an interaction. Hence, the tool is also useful for agent-based simulations that are useful in many areas. It is not open source but is free to use.

GameSalad [64], [65] is a game-authoring tool that provides a GUI to specify rules and actions for a game and requires zero knowledge of programming. Its library provides functions for sensing and physics and is also equipped with the behaviors of game objects known as actors. It is being used to teach basic concepts of computing as well as logic-based thinking. It is available for Mac and Microsoft Windows platforms. GameSalad is neither free nor an open-source language.

Blockly [66], [67] by Google is a web-based, free, and open-source VPL that makes computer programming fun by just dragging and connecting jigsaw blocks. These blocks are related to various programming constructs, including input/output, conditions, and loops. Blockly facilitates the development of new VPLs, and several VPLs based on Blockly for Games, Hobbyist programmers, IoT, Robotics, and other domains are available. It is equipped with most of the programming constructs of casual need.

BlockPy [68] is a Blockly and Scratch based visual programming environment that can generate Python code. In addition to providing visual scripting for casual coding by novice or hobbyist programmers, it is a powerful tool that can be used to solve data science problems. Unlike a typical VPL, BlockPy may display output in the form of histograms and plots. It supports several predefined data sets to work with. It is a web-based, open-source, and free tool.

Microsoft VPL [69] is a programming environment based on graphical data-flows. It is aimed at engaging hobbyist programmers as well as professionals. Furthermore, novice programmers can use it to learn programming, and experienced programmers can employ it for rapid prototyping. Microsoft VPL is a part of Microsoft Robotics Developers Studio (MRDS) used to develop software to guide robots. Figure 4 shows a simple visual program written using Microsoft VPL. Pocket Code [70] from Catrobat is a VPL for Android phones. It allows end-users to create their own games, animations, and interactive music videos. Inspired by the Scratch environment, in Pocket Code, games are developed through scripting command blocks (known as bricks) by placing them one after the other. For parallel execution, more than one script of bricks may be used. It is a free and open-source VPL.

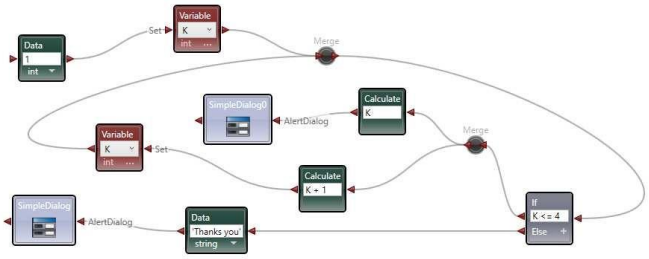

Figure 4: A simple visual program of Microsoft VPL

Lego MindStorms Software [71], [72] are toy robots used in academia for introductory robotics courses as well as in robotics-related research. The manufacturer provides free software for controlling robots in a visual programming environment. A programmer can drag-and-drop GUI objects, set their properties, and connect them in a sequence to control a robot. The software supports threads to provide parallelism in various robotic activities. Lego MindStorms software has been discontinued, but the older version is still available for download and program Lego robots.

VIPLE [73] was developed at Arizona State University and is referred to as ASU VIPLE. It is based on Microsoft VPL and is used to develop Robotics and IoT applications. It provides constructs like conditions, loops, and input/output, so that it may be used in teaching a programming fundamentals course [74]. It can be used to program finite state machines as well as for event-driven programming. VIPLE is not only partially open source but is also free to use.

Open Roberta Fraunhofer IAIS developed Open Roberta [75] in 2013 using a grant from Google. Open Roberta is a VPL for different robotic hardware platforms, including Lego’s EV3 and Arduino’s Bot’n Roll ONE A. It provides a cloud-based environment with a variety of sensors and motor controls. Open Roberta is platform-independent with a web-based interface. A visual program developed in Open Roberta is directly deployed on the target robot and, thus, cannot be modified manually on a desktop computer. It is free and open-source.

mBlock [76] is a free and open-source visual programming tool for kids to learn and visually program well-known robot kits. MakeBlocks, the developer of

mBlock, also promotes STEM education in the United Kingdom. It is based on the Scratch 2.0 offline version and is available for the Microsoft Windows and Mac-OS platforms.

Pencil Code [77] is also a web-based, free, and open-source block-based comprehensive programming environment. Though, it looks like a Turtle graphics tool but it has many additional features like support of input/output, loops, conditional-jumps, functions, arrays, and recursion. Therefore, unlike the typical Turtle graphics languages, Pencil Code may be considered a much better tool to learn the basics of programming.

StartLogo [78], [79] is another VPL developed at MIT to simulate agents moving around in a 3D environment. It is designed for educational purposes to model the behavior of decentralized systems. The latest version of StarLogo is StarLogo Nova, which provides a browser-based 3D visualization engine to process the block language, called ScriptBlocks. ScriptBlocks is based on a JavaScript blocks library, which can move tens of thousands of independent agents simultaneously. StarLogo is written in Java and hence can run on almost all platforms. Although, in general, StarLogo is not open source, however, its version OpenStarLogo is open source and free to use.

Alice [80]–[82] provides a drag-and-drop environment to prevent early programmers from making syntax errors. Novice programmers can learn object-oriented programming concepts using Alice by making 2D and 3D animations and games. Alice provides a virtual environment to create animations, stories, or games with characters controlled by instructions similar to textual programming languages. It is not a general-purpose programming environment but is helpful in learning the basics of programming.

Analytica [72] is a visual software meant to develop and analyze quantitative decision models like in spreadsheets. However, its interface and functionality are better organized as compared to simple spreadsheet software. It facilitates users to quickly evaluate uncertainties and risks through its efficient Monte-Carlo simulation. It also provides a rich set of operations and functions used for general-purpose declarative programming. In Analytica, users may create programs as models by visually adding and linking visual elements or nodes. It provides a construct called Intelligent Arrays to manage multidimensional data easily. Analytica is only available for Microsoft Windows operating system and is a commercial software, which also provides a limited free version to develop and analyze medium-sized models. It is not open source.

Beetle Blocks like a LOGO's Turtle, which can move on a 2D surface, a Beetle is an object that can fly too. So, Beetle Blocks [83] may be considered a 3D version of LOGO. It is based on Scratch and can be used for 3D designing and printing. A Beetle object, that can be moved in 3D through visual instructions, can place 3D shapes along its path or can extrude a tube. Beetle Blocks is in the development stage and is available as an alpha release. It is a browser-based open-source tool.

LabVIEW [84]–[87] is a proprietary (non-open-source) software providing graphical programming syntax to interact with any hardware device to acquire and visualize its data (input, output, gain, current, voltage, etc.). Basically, LabVIEW creates virtual instruments (VI) for data acquisition and visualization, e.g., oscilloscopes and multimeters. LabVIEW provides the option to create a subVI to make a custom interface block. It also provides options to create functions to extend the functionality of LabVIEW for programming a VI. LabVIEW does not generate code in any textual language. It provides options to do generic programming, but it is mainly used to build specific programs called VIs. It is not free software.

Dynamo [88] is an open-source software platform for information modeling that provides a visual interface to develop programs. It is freely available for academic purposes only. The development of Dynamo started in the late 1950's at the MIT Computing Center, and it has evolved since then. Initially, several versions of Dynamo were developed using ALGOL, FORTRAN, and Assembly language; the current version is based on REVIT - a modeling software. The use of Dynamo is limited to modeling industrial dynamics, population and resources in urban planning. It is textual language-independent, and it can be extended using Python scripts and importing OS libraries. The current version is supported on the Windows platform only.

Kodu [89], [90] was developed by Microsoft for Windows and Xbox-360 as a visual game development software. It is independent of any textual programming language. It uses icon-based programming, where programs are composed of pages that are further decomposed into rules, conditions, and actions. It supports many high-level real-world primitives, including vision, color, and collision. It also provides an interactive terrain editor. Although Kodu is freely available for Microsoft Windows however the Xbox 360's version is commercial. It is not open source.

ANALYSIS AND COMPARISON OF VPLS

This section presents one of the main contributions of the study, which is the analysis and comparison of VPLs. In particular, firstly, the characteristics used for the analysis and comparison of the VPLs are presented. These characteristics are subsequently used to assess the possessiveness of the characteristic in VPLs, which is used to evaluate and compare the VPLs.

Many characteristics are binary in nature, i.e., a VPL either has that characteristic or it does not. For a binary characteristic, the score is either 1 or 0. In contrast, some characteristics are not binary in nature and thus may be assigned a value between 0 and 1. These characteristics are shown in Table 3. We now discuss these characteristics and the scores assigned to them for the chosen set of VPLs. Furthermore, the characteristics of the VPLs are categorized into two broader types named as

- **Extrinsic**: the characteristics associated to VPLs but not as their part.
- **Intrinsic** or completeness: the characteristics associated to VPLs as their part, and VPL programs can be written according to these characteristics. These are actually the idea of the strength of the VPL.

Table 3: Characteristics of VPLs used in this paper.

Sr. no.	Characteristics	Binary?
	Extrinsic	
1	Open source (OS)	✓
2	Free (FR)	✓
3	Language independence (LI)	✓
4	Platform independence (PI)	✗
5	Textual code generation (TC)	✗
6	Extensibility (EX)	✗
7	Genericness (GP)	✓
	Completeness	
8	Arrays (AY)	✓
9	Modular approach (MA)	✓
10	Object-oriented (OO)	✓
11	File I/O (IO)	✓
12	Debugging (DG)	✓
13	Exception handling (EH)	✓
14	Threads (TH)	✓

Extrinsic characteristics of VPLs
The extrinsic characteristics of VPLs are the following:

1. Open Source: Open source software (OSS) movement started in the late 1990s to provide free and community-controlled software to the end-users, along with source code to counter software monopoly [91]. Nowadays, the OSS is becoming a mainstream philosophy because even well established large IT enterprises, such as Google, Oracle, and Facebook, are making their proprietary software open source [92]. This has enabled community collaboration on a large scale to improve and extend the functionality of OSS. Generally, OSS tends to have fewer bugs because a large and vigilant community of developers may quickly fix a newly-discovered bug [91]. Therefore, we have selected open source as a characteristic for evaluating VPLs. Open source VPLs may be extended and enhanced by the collective effort of the open-source community. Furthermore, multiple variants of the existing open-source VPLs may be designed and developed. Open source is a binary characteristic that a VPL either possesses or it does not. A column in Table 4 shows the scores for the VPLs for this characteristic.
2. Free: Many open source projects are available to the public free of cost. The companies developing free and open-source software usually earn money through services and customization of such software. Free software can easily be accessed by a large number of people with diverse backgrounds, such as underprivileged students, small non-profit organizations, and academic institutions, particularly in underdeveloped countries. Free is a binary characteristic that can be assigned a score of 0 or 1. Table 4 lists VPLs' scores with 1 meaning free and 0 otherwise.

3. Language independence: A VPL should not be based on a specific textual language such as Java or C++. Instead, a VPL should use generic programming language constructs to cater commonalities of most of the textual programming languages. Language independence has two advantages. Firstly, programmers of a VPL could acquire a good insight into the fundamentals of programming instead of getting bogged down into the nitty-gritty of a specific textual programming language. Secondly, a language-independent VPL could generate code for several textual languages, which might not be feasible for a VPL based on a specific textual language. Table 4 lists scores of the language independence characteristics. A language is assigned a score of 1 if its visual interface is not coupled with any of the textual programming languages. Otherwise, it is assigned a score of 0.
4. Platform independence: In order to make a VPL accessible to the masses, it should support as many development platforms as possible. Table 4 lists scores for the platform independence characteristic of the VPLs. The well-known development platforms for desktop computers are Windows, Mac and Linux. We consider a VPL fully platform-independent when it can be programmed on all three well-known platform systems, which may be achieved either through the availability of binaries for each on these platforms or a VPL being Web or JVM based. A fully platform-independent language is assigned a score of 1. In case a VPL can only be used in a subset of the three platforms then it is assigned a score 0.33 or 0.67 depending upon its availability for 1 or 2 platform, respectively.
5. Textual code generation: A VPL environment should facilitate the generation of code in existing well-known textual languages such as C/C++, Java, JavaScript, and Python. It is because textual code may be helpful to amateur programmers to experiment with and learn the relevant textual language without the time-consuming effort of manually typing a syntactically correct program in a textual language. Thus, a trainee programmer, who is a school going child or an undergraduate student in early semesters, could be quickly introduced to several textual programming languages using the code generation capabilities of a VPL. As textual languages have been around since long, they possess a degree of maturity with fine-tuned performance for speed and RAM usage due to advancements in compiler technology [93]. Generation of textual code by a VPL is like having the best of both worlds. That is, a programmer can develop program code using an easy to use GUI and at the same time enjoy the performance of a textual language, provided the automatically translated textual code is

reasonable enough like written by a professional programmer. The characteristic of textual code generation is not binary. Thus, it can be assigned any score between 0 and 1 depending on the number of textual languages in which a VPL's visual code can be translated to.

Table 4: Extrinsic characteristics scores: For binary characteristics score either 1 or 0. For Language Independence score is [0-3], for Platform Independence score is [0-10] and for Extensibility score is [0-1].

Sr.	VPL	Open Source	Free	Language Independent	Platform Independence	Textual Code Generation	Extensibility	Genericness
1	Flowgorithm	0	1	1	1	10	1	1
2	Progranimate	0	1	1	3	6	1	1
3	Flowchart Interpreter	0	1	0	1	1	1	1
4	Raptor	1	1	1	2	5	1	1
5	Larp	0	1	1	1	0	1	1
6	Visual Logic	0	0	1	1	2	1	1
7	DRAKON	0	1	1	3	10	1	1
8	Iconic Programmer	0	1	1	3	3	1	1
9	devFlowcharter	1	1	0	1	3	1	1
10	Marten (Prograph)	0	1	1	1	0	1	1
11	App Inventor for Android	1	1	1	2	1	1	1
12	B#	0	0	0	1	3	1	1
13	The SFC Editor	0	1	0	1	0	1	1
14	BACCII/BACCII++	1	0	1	1	3	0.5	1
15	FLINT	0	0	0	1	0	1	0
16	SIVIL	0	0	0	3	0	1	0
17	Scratch	0	1	1	3	0	0.5	1
18	Snap	0	1	1	3	0	1	1
19	Tynker	0	1	1	3	1	1	0
20	Stencyl	0	1	1	3	0	1	0
21	Agent Sheets	0	0	1	2	0	1	0
22	CODE	0	0	0	3	0	1	0
23	EToys	0	1	1	3	0	1	0
24	GameSalad	0	0	1	2	0	1	0
25	Blockly	1	1	1	3	5	0.8	1
26	Touch Develop	0	1	1	3	1	1	1
27	BlockPy	0	1	0	3	1	1	1
28	Microsoft VPL	0	0	1	1	0	1	1
29	Lego MindStorms Software	0	0	1	1	2	1	1
30	VIPLE	1	1	1	1	0	1	1
31	Open Roberta	1	1	1	3	1	1	1
32	mBlock	1	1	1	3	0	0.5	0
33	Pencil Code	1	1	1	3	0	1	1
34	Beetle Blocks	1	1	1	3	0	1	0
35	Alice	0	1	1	1	0	1	0
36	Kudo	0	0	1	1	0	1	0
37	StarLogo TNG	0	1	1	3	0	1	0
38	Dynamo	1	1	1	1	0	1	0
39	LabVIEW	0	0	1	3	0	1	1
40	Analytica	0	0	0	1	0	1	0

We can assign a maximum score of 1 to a VPL if its visual code can be translated into 10 or more textual languages.

6. Extensibility: Extensibility refers to the property of software to take future enhancements into consideration by keeping the design flexible. A VPL should be extensible with respect to three aspects: (1) the ability to add new visual elements (visual design constructs), (2) the ability to change and enhance the meaning of an existing visual element, and (3) inclusion of an additional textual programming language for code generation.

Visual elements are dragged and dropped for graphically implementing an algorithm. Extensibility demands the inclusion of new programming constructs in a VPL by introducing additional visual elements. Interpretation of a visual element should also be extensible so that interpretation could be improved with time as the VPL matures. We should be able to enhance the quality of code generated by a VPL environment. Finally, the code generation support for additional textual languages should also be extensible. The scores for the extensibility characteristic are shown in Table 4. Although many VPLs are open source but are not easy to extend without spending significant time in understanding and modifying their source code. Therefore, we have given an extensibility score to a VPL when its developers have made an effort by providing documentation for extensibility and have specified steps that can be used to extend a VPL.

7. Genericness: Literature survey reveals that many VPLs were designed to develop toy programs, or they are domain-specific [41], [42], [46]. For example, Scratch is used to develop simple stories, games, and animations, and Alice uses animations to teach the basics of programming [41], [46]. In most of the existing VPLs, it is not feasible to write a complex program of a few thousand instructions. Thus, it is desirable that a VPL is general-purpose programming language and allows the implementation of enterprise-level software development. For example, a VPL environment should allow the development of complex video games as well as the implementation of solutions of complicated

scientific problems. Table 4 shows the assignment of scores to the VPLs for this characteristic. We assigned a score of 1 if a VPL is general purpose and can be used to visually code at least basic computational logic such as a sorting algorithm. Otherwise, a score of 0 is given to the VPL.

Intrinsic characteristics of VPLs

Most of the existing VPLs lack basic constructs required for the development of general-purpose computer programs or for implementing enterprise-level software [41], [42], [46]. We consider a programming language complete if it has all the basic constructs for software development. The following features are desirable for a complete VPL.

1. Arrays: Many computer programs require storage and retrieval of large amounts of related data in RAM for fast and random access, e.g., a collection of books, country names, subject of an undergraduate course, and series of numeric values. Almost all textual programming languages support arrays to store such data. Completeness requires that a VPL has support for arrays.

2. Modularity and Object-Oriented support: As the size of a program grows, it becomes difficult to manage it as a single programming block. Thus, to manage large programs, modular support should be available in a VPL. Similarly, to create easy to reuse and extended components, object-oriented support is desirable in a VPL.

3. File I/O: In a large number of applications, file I/O is used in order to make programs independent from continuous user interaction through the console. This facilitates batch execution of a program that entails creating a program binary once while using the same binary for a set of input files to generate a set of output files without user intervention. Thus, in order for a VPL to allow programming of several real-world tasks, it must support file I/O.

4. Debugging and Exception handling: To develop robust programs, a VPL should provide debugging and exception handling capabilities. A visual program must have the flow information embedded in it that could be used while debugging the program to identify logical errors in it. Similarly, to catch runtime exceptions, proper structured exception handling facilities needs to also be part of a VPL.

5. Threads: Many modern applications require simultaneously running multiple pieces of code (i.e., tasks) within a program. For instance, a task may be handling an I/O channel while another may be completing a lengthy background procedure. Similarly, games with more than one character also require different logic for parallel handling of each character independent of the other. To enable near simultaneous execution of different components of a program, threads are essential. Thus, to be complete, a VPL should support threads.

LIMITATIONS, FUTURE DIRECTIONS AND CONCLUSIONS

In this paper, we have presented the most comprehensive survey of visual programming languages. This section presents the limitations, future directions and conclusion of our research and the future work, we are considering on the basis of it.

Table 5: Completeness scores: Each characteristic of completeness is binary and hence score either 1 or 0.

Sr.	VPL	Arrays	Modular	Object Oriented	File I/o	Debugging	Exception Handling	Threads
1	Flowgorithm	1	1	0	0	1	0	0
2	Progranimate	1	0	0	0	1	0	0
3	Flowchart Interpreter	0	1	0	0	0	0	0
4	Raptor	1	1	1	1	1	0	0
5	Larp	1	1	0	1	1	0	0
6	Visual Logic	1	1	0	1	1	0	0
7	DRAKON	1	1	1	1	0	0	1
8	Iconic Programmer	0	0	0	0	0	0	0
9	devFlowcharter	1	1	0	0	0	0	0
10	Marten (Prograph)	1	1	1	1	1	0	0
11	App Inventor for Android	1	1	1	0	1	1	0
12	B#	0	0	0	0	1	0	0
13	The SFC Editor	0	1	0	0	0	0	0
14	BACCII/BACCII++	0	1	1	0	1	0	0
15	FLINT	0	1	0	0	1	0	0
16	SIVIL	0	0	0	0	1	0	0
17	Scratch	1	1	0	0	1	0	1
18	Snap	1	1	1	0	0	0	1
19	Tynker	1	1	0	0	1	0	1
20	Stencyl	1	1	0	0	1	0	1
21	Agent Sheets	1	0	1	0	0	0	1
22	CODE	0	1	0	0	1	0	0
23	EToys	0	0	1	0	1	0	1
24	GameSalad	1	0	1	0	0	0	1
25	Blockly	1	1	0	0	1	0	0
26	Touch Develop	1	1	1	0	1	0	1
27	BlockPy	1	1	0	0	1	0	0
28	Microsoft VPL	1	1	0	0	1	0	1
29	Lego MindStorms Software	1	1	0	1	1	0	1
30	VIPLE	0	0	0	0	0	0	1
31	Open Roberta	0	1	0	0	0	0	1
32	mBlock	1	1	0	0	0	0	1
33	Pencil Code	1	1	1	0	1	0	0
34	Beetle Blocks	1	1	0	0	0	0	1
35	Alice	1	1	1	0	0	0	1
36	Kudo	0	0	1	0	1	0	1
37	StarLogo TNG	1	1	0	0	0	0	0
38	Dynamo	1	1	0	1	0	0	0
39	LabVIEW	1	1	1	1	1	0	1
40	Analytica	1	1	0	1	1	1	0

Limitations

- Potentially incomplete VPLs list: The paper has the largest collection of VPLs ever included in single study. Furthermore, to identify these 40 VPLs a systematic procedure is employed and each characteristic is collected using cross-validation. However, there may be some VPLs that were not identified during our two years old search process. Thus, we encourage the readers to identify any additional VPL and compute the ranking of the newly identified VPL, in various contexts.
- Exclusion due to lack of documentation: One reason to exclude few VPLs from the ranking procedure is the absence of sufficient information. That is, a VPL is excluded for which multiple researchers while working independently, were unable to find the necessary information to extract its characteristics and execute the ranking procedure. Such a VPL, might still be beneficial in some cases, however, we are impuissant to compute its ranking due to the lack of documentation availability.

Future Directions

Based upon the data acquired in the various tables presented in the paper text, we are working on a novel ranking method to rank the VPLs and similar stuff available in the public domain. We will have a survey about the significance of each of the characteristics, and our ranking algorithms will use both the score and the significance of each characteristic. We are also moving to develop a state-of-the-art VPL development and useful tool.

CONCLUSION

VPLs reduce the effort of writing computer programs by providing visual elements. In this paper, we have made three key contributions which are summarized below. Firstly, a systematic procedure is employed to identify a collection of VPLs and gather their characteristics using cross-validation. To the best of our knowledge, this is the largest ever collection of VPLs included in a single study. Secondly, a set of characteristics are introduced to assess and computer VPLs. Thirdly, a concise overview of these many VPLs is presented, which can be a valuable reference for the researchers and developers interested in VPLs. Finally, a comparison of the VPLs is presented.

ACKNOWLEDGMENTS

The authors thank Hafiz Muhammad Danish, Arooba Liaqat Ali, Hafsa Saleem, and Maria Akber for helping in the collection and verification of data. Finally, we thank Iqbal Ali Azhar for proofreading the initial draft of the paper and providing feedback.

Note: This paper is based upon the research work conducted during Ph.D. Therefore, a large part of the text is taken from the PhD thesis of the first author.

CREDIT AUTHOR STATEMENT

Both **Muhammad Idrees** and **Faisal Aslam** equally worked on the Conceptualization, Methodology, and writing of original draft, whereas the data in tables and diagrams are mainly gathered and formulated by the **Muhammad Idrees** under the supervision of **Faisal Aslam**.

COMPLIANCE WITH ETHICAL STANDARDS

It is declared that all authors don't have any conflict of interest. Furthermore, informed consent was obtained from all individual participants included in the study.

REFRENCES

[1] A. Orlowska, C. Chrysoulas, Z. Jaroucheh, X. Liu, "Programming Languages: A Usage-based Statistical Analysis and Visualization." *4th International Conference on Information Science and Systems*, Edinburgh, UK, pp. 143-148, 2021.

[2] E. C. Harel and E. R. McLean, "The effects of using a nonprocedural computer language on programmer productivity," *MIS Quarterly*, vol. 9, pp. 109–120, 1985.

[3] M. Idrees, F. Aslam, K. Shahzad and S. M. Sarwar, "Towards a universal framework for visual programming languages," *Pakistan Journal of Engineering and Applied Sciences*, vol. 23, pp. 55-65, 2018.

[4] B. A. Myers, A. J. Ko, and M. M. Burnett, "Invited research overview: end-user programming," Conference on Human Factors in Computing Systems, Quebec, Canada, 2006, pp. 75–80.

[5] J. M. Chin, M. H. Chin, and C. Van Landuyt, "A string search marketing application using visual programming," *The E-Journal of Business Education & Scholarship of Teaching*, vol. 7, p. 46-59, 2013.

[6] B. A. Myers, "Taxonomies of visual programming and program visualization," *Journal of Visual Languages & Computing*, vol. 1, pp. 97–123, 1990.

[7] D. Ingalls, S. Wallace, Y.-Y. Chow, F. Ludolph, and K. Doyle, "Fabrik: a visual programming environment," *ACM SIGPLAN Notices*, vol. 23, pp. 176–190, 1988.

[8] "Squeak," http://wiki.squeak.org/squeak/1227, accessed: 2016-08-16.

[9] D. D. Hils, "Visual languages and computing survey: Data flow visual programming languages," *Journal of Visual Languages & Computing*, vol. 3, pp. 69–101, 1992.

[10] M. Boshernitsan and M. S. Downes, "Visual programming languages: A survey," Computer Science Division, University of California, USA, 2004.

[11] M. Najork, "Visual programming in 3-d," *Dr Dobb's Journal-Software Tools for the Professional Programmer*, vol. 20, pp. 18–33, 1995.

[12] R. Singh, "Mosgen: A visual programming tool for mobile services," MS Thesis, Department of Computer Science and Electrical Engineering, Lulea University of Technology, Sweden, 2008.

[13] A. S. Scott, "Using flowcharts, code and animation for improved comprehension and ability in novice programming," PhD thesis, Faculty of Advanced Technology, University of Glamorgan, 2011.

[14] S. Xinogalos, "Using flowchart-based programming environments for simplifying programming and software engineering processes," *Global Engineering Education Conference (EDUCON)*, Berlin, Germany, 2013, pp. 1313–1322.

[15] D. Hooshyar, T. Ma´ıen, and M. Masih, "Flowchart-based programming environments aimed at novices," *International Journal of Innovative Ideas*, vol. 13, pp. 52–62, 2013.

[16] R. Vanwersch, K. Shahzad, K. Vanhaecht, P. Grefen, L. Pintelon, G. van Merode, and H. Reijers, "Business process redesign in health care: towards a comprehensive methodological framework," *International Journal of Care Pathways*, vol. 16, pp. 48–48, 2012.

[17] M. D´ıaz and J. Luis, "Remgrafee tool: Herramienta para el estudio de los sistemas basados en reglas mediante grafos rete." 2016. https://ruidera.uclm.es/xmlui/handle/10578/10278

[18] "Flowgorithm," http://www.flowgorithm.org/, accessed: 2022-04-10.

[19] E. Perrin, S. Linck, and F. Danesi, "Algopath: A new way of learning algorithmic," *Fifth International Conference on Advances in Computer-Human Interactions*, Valencia, Spain, 2012, pp. 291–296.

[20] "Logic of algorithms for resolution of problems (larp)," http://larp.marcolavoie.ca/en/default.htm, accessed: 2022-04-10.

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

[22] P. Nikunja Swain, "Raptor-a vehicle to enhance logical thinking," *Journal of Environmental Hazards*, vol. 7, pp. 353–359, 2013.

[23] "Raptor, a flowchart-based programming environment," http://raptor.martincarlisle.com/, accessed: 2016-08-10.

[24] A. Scott, M. Watkins, and D. McPhee, "Progranimate-a web enabled algorithmic problem solving application," *International Conference on E-Learning, E-Business, Enterprise Information Systems, and E-Government*, Nevada, USA, 2008, pp. 498–508.

[25] A. Scott, M. Watkins, D. McPhee, "E-learning for novice programmers; a dynamic visualization and problem solving tool," 3rd International Conference on Information and Communication Technologies: From Theory to Applications, Damascus, Syria, 2008, pp. 1–6.

[26] "Progranimate," http://www.progranimate.com/, accessed: 2016-08-16.

[27] "Flowchart interpreter," http://vardanyan.am/fi/, accessed: 2016-08-10.

[28] "Visual logic," http://www.visuallogic.org/, accessed: 2022-04-10.

[29] S. Mitkin, "Drakon: The human revolution in understanding programs," http://drakon-editor.sourceforge.net/DRAKON.pdf, 2011.

[30] S. Chen and S. Morris, "Iconic programming for flowcharts, java, turing, etc," *ACM SIGCSE Bulletin*, vol. 37, pp. 104–107, 2005.

[31] "Sourceforge devflowcharter," http://devflowcharter.sourceforge.net/, accessed: 2022-04-10.

[32] R. Roberts, Google App Inventor. *Packt Publishing* Ltd, 2011.

[33] R. N. Horspool and N. Tillmann, *TouchDevelop: programming on the go*, Springer, The Netherland, 2013.

[34] N. Tillmann, M. Moskal, J. de Halleux, M. Fahndrich, and T. Xie, "Engage your students by teaching computer science using only mobile devices with touchdevelop," *International Conference in Software Engineering Education and Training*, Nanjing, China, 2012, pp. 87–89.

[35] "Touch develop," https://www.touchdevelop.com/, accessed: 2016-0816.

[36] "Touch develop export," https://www.touchdevelop.com/docs/export-toapp, accessed: 2016-08-16.

[37] M. Koorsse, C. Cilliers, and A. Calitz, "Programming assistance tools to support the learning of it programming in south african secondary schools," *Computers & Education*, vol. 82, pp. 162–178, 2015.

[38] J. Greyling, C. Cilliers, and A. Calitz, "B#: The development and assessment of an iconic programming tool for novice programmers," *7th International Conference on Information Technology Based Higher Education and Training*, Ultimo, Australia, 2006, pp. 367–375.

[39] B. Calloni and D. Bagert, "Iconic programming in baccii vs. textual programming: which is a better learning environment?" *ACM SIGCSE Bulletin*, vol. 26, pp. 188–192, 1994.

[40] B. Calloni, D. Bagert, and H. Haiduk, "Iconic programming proves effective for teaching the first year programming sequence," *ACM SIGCSE Bulletin*, vol. 29, pp. 262–266, 1997.

[41] M. Resnick, J. Maloney, A. Monroy-Hernandez, N. Rusk, E. Eastmond,´ K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman et al., "Scratch: programming for all," *Communications of the ACM*, vol. 52, pp. 60–67, 2009.

[42] S. Fincher, S. Cooper, M. Kolling, and J. Maloney, "Comparing alice, greenfoot & scratch," 41st ACM technical symposium on Computer science education, Wisconsin, USA, 2010, pp. 192–193.

[43] J. H. Maloney, K. Peppler, Y. Kafai, M. Resnick, and N. Rusk, "Programming by choice: urban youth learning programming with scratch", *ACM SIGCSE Bulletin*, vol. 40, pp. 367-371, 2008.

[44] "Scratch," https://scratch.mit.edu/, accessed: 2016-08-10.

[45] M. Rizvi and T. Humphries, "A scratch-based cs0 course for at-risk computer science majors," *International Conference on Frontiers in Education, Seattle*, USA, 2012, pp. 1–5.

[46] B. Harvey, D. D. Garcia, T. Barnes, N. Titterton, O. Miller, D. Armendariz, J. McKinsey, Z. Machardy, E. Lemon, S. Morris et al., "Snap!(build your own blocks)," *45th ACM technical symposium on Computer science education*, Atlanta, USA, 2014, pp. 749–749.

[47] D. Kumar, "Digital playgrounds for early computing education," *ACM Inroads*, vol. 5, pp. 20–21, 2014.

[48] F. J. Garcia-Penalvo, A. Rees, J. Hughes, I. Jormanainen, T. Toivonen, and J. Vermeersh, "A survey of resources for introducing coding into schools," *Fourth International Conference on Technological Ecosystems for Enhancing Multiculturality*, Salamanca, Spain, 2016, pp. 19-26.

[49] U. Ziegler and T. Crews, "An integrated program development tool for teaching and learning how to program," *ACM SIGCSE Bulletin*, vol. 31, pp. 276–280, 1999.

[50] T. Materson and R. Meyer, "Sivil: a true visual programming language for students," *Journal of Computing Sciences in Colleges*, vol. 16, pp. 74–86, 2001.

[51] "Code dot org," https://code.org/about, accessed: 2016-08-16.

[52] F. Kalelioglu, "A new way of teaching programming skills to k-12˘ students: Code. org," *Computers in Human Behavior*, vol. 52, pp. 200– 210, 2015.

[53] H. K. Bhargava, S. Sridhar, and C. Herrick, "Beyond spreadsheets: tools for building decision support systems," *Computer*, vol. 32, pp. 31– 39, 1999.

[54] "Sfc website," http://watts.cs.sonoma.edu/SFC/, accessed: 2016-08-10.

[55] J. M. Rouly, J. D. Orbeck, and E. Syriani, "Usability and suitability survey of features in visual ides for non-programmers," *5th Workshop on Evaluation and Usability of Programming Languages and Tools*, Portland, USA, 2014, pp. 31–42.

[56] R. Sneyd, *Stencyl Essentials*, *Packt Publishing Ltd*, 2015.

[57] "Stencyl," http://www.stencyl.com/, accessed: 2016-08-16.

[58] A. Kay, "Squeak etoys authoring & media," Viewpoints Research Institute, 2005.

[59] B. Freudenberg, Y. Ohshima, and S. Wallace, "Etoys for one laptop per child." *C5*, vol. 9, pp. 57–64, 2009.

[60] A. Repenning, "Agentsheets®: An interactive simulation environment with end-user programmable agents," *Interaction*, 2000.

[61] "Agentsheets," http://www.agentsheets.com/, accessed: 2016-08-16.

[62] A. Repenning and T. Sumner, "Agentsheets: A medium for creating domain-oriented visual languages," *Computer*, vol. 28, pp. 17– 25, 1995.

[63] A. Repenning, "Agentsheets: a tool for building domain-oriented visual programming environments," *International Conference on Human factors in computing systems*, Amsterdam, The Netherlands, 1993, pp. 142–143.

[64] M. DeQuadros, GameSalad Beginner's Guide. *Packt Publishing Ltd*, 2012.

[65] K. Roy, W. C. Rousse, and D. B. DeMeritt, "Comparing the mobile novice programming environments: App inventor for android vs. gamesalad," International Conference on Frontiers in Education, Seattle, Washington, 2012, pp. 1–6.

[66] N. Fraser et al., "Blockly: A visual programming editor," URL: https://code. google. com/p/blockly, 2013.

[67] A. Marron, G. Weiss, and G. Wiener, "A decentralized approach for programming interactive applications with javascript and blockly," *2nd international confrence on Programming systems, languages and applications based on actors, agents, and decentralized control abstractions*, Tucson, Arizona, USA, 2012, pp. 59–70.

[68] A. C. Bart, E. Tilevich, C. A. Shaffer, and D. Kafura, "Position paper: From interest to usefulness with blockpy, a block-based, educational environment," *IEEE workshop on Blocks and Beyond Workshop (Blocks and Beyond)*, Atlanta, Gergia, USA, 2015, pp. 87–89.

[69] "Microsoft vpl," https://msdn.microsoft.com/enus/library/bb483088.aspx, accessed: 2022-05-16.

[70] W. Slany, "Tinkering with pocket code, a scratch-like programming app for your smartphone," *Proceedings of the Constructionism*, Vienna, Austria, 2014, pp. 1-2.

[71] F. Klassner and S. D. Anderson, "Lego mindstorms: Not just for k-12 anymore," *IEEE Robotics & Automation Magazine*, vol. 10, pp. 12–18, 2003.

[72] P. B. Lawhead, M. E. Duncan, C. G. Bland, M. Goldweber, M. Schep, D. J. Barnes, and R. G. Hollingsworth, "A road map for teaching introductory programming using lego mindstorms robots," *ACM SIGCSE Bulletin*, vol. 35, pp. 191–201, 2002.

[73] "Viple," http://neptune.fulton.ad.asu.edu/VIPLE/, accessed: 2016-08-16.

[74] Y. Chen and G. De Luca, "Viple: Visual iot/robotics programming language environment for computer science education," IEEE Internatnaionl Synposium on Parallel and Distributed Processing, Chicago, USA, 2016, pp. 963–971.

[75] M. Ketterl, B. Jost, T. Leimbach, and R. Budde, "Tema 2: Open robertaa web based approach to visually program real educational robots," *Tidsskriftet Læring og Medier (LOM)*, vol. 8, pp. 1-22, 2016.

[76] "mblock," http://www.mblock.cc/, accessed: 2022-05-16.

[77] D. Bau, D. A. Bau, M. Dawson, and C. Pickens, "Pencil code: block code for a text world," *14th International Conference on Interaction Design and Children*, Boston, USA, 2015, pp. 445–448.

[78] M. Resnick, "Starlogo: An environment for decentralized modeling and decentralized thinking," *International Conference Companion on Human factors in computing systems*, Vancouver, Canada, 1996, pp. 11–12.

[79] K. Wang, C. McCaffrey, D. Wendel, and E. Klopfer, "3d game design with programming blocks in starlogo tng," *7th International Conference on Learning Sciences*, Bloomington, USA, 2006, pp. 1008–1009.

[80] M. Conway, S. Audia, T. Burnette, D. Cosgrove, and K. Christiansen, "Alice: lessons learned from building a 3d system for novices," *International SIGCHI conference on Human Factors in Computing Systems*, The Hague, The Netherlands, 2000, pp. 486–493.

[81] S. Cooper, W. Dann, and R. Pausch, "Alice: a 3-d tool for introductory programming concepts," *Journal of Computing Sciences in Colleges*, vol. 15, pp. 107-116, 2000.

[82] "Alice," http://www.alice.org/, accessed: 2022-04-10.

[83] B. Romagosa Carrasquer, "From the turtle to the beetle," Master's thesis, Universitat Oberta de Catalunya, 2016.

[84] J. Travis and J. Kring, *LabVIEW for everyone: graphical programming made easy and fun*, Prentice-Hall, 2007.

[85] G. W. Johnson, *LabVIEW graphical programming*, Tata McGraw-Hill Education, 1997.

[86] L. K. Wells and J. Travis, *LabVIEW for everyone: graphical programming made even easier*, Prentice-Hall, Inc., 1996.

[87] "What is labview?" http://www.ni.com/en-lb/shop/labview.html, accessed: 2017-09-01.

[88] "Open source graphical programming for design," http://dynamobim.org/, accessed: 2017-09-01.

[89] M. MacLaurin, "Kodu: end-user programming and design for games," *4th International conference on foundations of digital games*, Orlando, USA, 2009, pp. 1-2.

[90] M. B. MacLaurin, "The design of kodu: A tiny visual programming language for children on the xbox 360," ACM *SIGPLAN Notices*, vol. 46, pp. 241–246, 2011.

[91] W. Scacchi, "Understanding open source software evolution 181," *Software Evolution and Feedback: Theory and Practice*, vol. 1, pp. 181–206, 2006.

[92] A. Sharma, "Open source software–breaking the commercial myths," *International Journal of Advancements in Technology*, vol. 3, pp. 1-5, 2012.

[93] A. V. Aho, R. Sethi, and J. D. Ullman, *Compilers: principles, techniques, and tools*, Addison-wesley Reading, 2007, vol. 2.