Comparative analysis of 6 programming languages based on readability, writability and reliability

Zahin Ahmed
Department of Electrical and Computer Engineering
North South University
Dhaka, Bangladesh
zahin.ahmed@northsouth.edu

Farsihta Jayas Kinjol
Department of Electrical and Computer Engineering
North South University
Dhaka, Bangladesh
farishta.jayas@northsouth.edu

Ishrat Jahan Ananya
Department of Electrical and Computer Engineering
North South University
Dhaka, Bangladesh
ishrat.jahan16@northsouth.edu

Sabbir Hasan
Department of Electrical and Computer Engineering
North South University
Dhaka, Bangladesh
sabbir.hasan43@northsouth.edu

Bitan Debnath
Department of Electrical and Computer Engineering
North South University
Dhaka, Bangladesh
bitan.debnath@northsouth.edu

Md. Shahriar Karim
Department of Electrical and Computer Engineering
North South University
Dhaka, Bangladesh
shahriar.karim@northsouth.edu

***Abstract*—Since their inception, the development of programming languages had mostly been geared towards improving efficiency. However, in recent years, a lot of the attention has shifted towards making the languages readable and writable, while also improving reliability. These factors affect how many new users start to use a particular language, and how many experienced programmers continue to use it reliably in real applications. Hence, in this research, we have compared the readability, writability and reliability of six mainstream programming languages based on their characteristics. To verify the actuality of our findings, a survey was done to see how programmers and non-programmers judge the languages based on code snippets. Finally, experiments were carried out to see if there is a relation between the efficiency of a language and its readability, writability.**

Keywords—programming languages, readability, writability, reliability, efficiency.

I. Introduction

A programming language is a formal language used to communicate with machines such as computers. Programming languages consist of a set of instructions, usually called the "syntax", and rules on how the syntax can be used in combination with each other. The first programming language "Plankalkül" was designed by German scientist Konrad Zuse, however it was never implemented[1]. Short Code, one of the first pseudocode languages, was developed by John Mauchly in 1949. Soon after, FORTRAN became the first high level programming language that was commercially implemented in 1954 by John Backus[1]. Consequently, many languages have been developed over the years for many different purposes. Some languages have been designed to be suitable for complex computations, while others cater to other domains such as Artificial Intelligence. There are two major categories of programming languages: Imperative languages and Declarative languages. Imperative languages such as C, C++, and Java are more commonly used by general people and are often called general purpose languages as well. Declarative languages such as functional languages (LISP) have specific use cases.

While most imperative programming languages have some basic similarities, the difference usually lies in syntax, how the syntax is converted to machine language, efficiency, closeness to machine language (high level vs low level) etc. In recent years, a lot of emphasis has been given on making languages more readable and writable for programmers. Languages such as Python, R, Julia etc. are much higher level compared to the languages they are built on (mostly C, C++, S and Scheme) [2], [3], and they were developed with the aim of higher abstraction and making the whole process of coding easier and more convenient[4]. Indeed, the readability and writability of code holds great importance to all types of users, starting from non-programmers, novice programmers or even experienced programmers. In today's world, almost everyone has to deal with some type of programming or code. People with no experience in coding will not be comfortable in reading or understanding code that is complicated or unable to express its purpose clearly. Similarly, novice programmers will face a huge learning curve if the syntax and rules of a language are not orthogonal and simple[5]. In this case both readability and writability are important, as they are trying to learn the syntax while also implement it themselves. Finally, experienced programmers usually deal with code that is implemented and used in industry. Such codebases are maintained over long periods of time, and the programmer(s) dealing with the code changes often. Hence, not only are readability and writability extremely important here (for the programmers to be able to understand and extend/modify the code as needed), but also reliability comes into play. Reliability of a programming language ensures that the code behaves the way it is supposed to, and carries out its purpose, at all times. This is very important in industry-level programming where the smallest of unforeseen circumstances can cause huge losses.

Hence, the aim of this research is to compare the readability, writability and reliability of 5 commonly used imperative programming languages, which are C, C++, Java, JavaScript, Python, and 1 functional language, which is R. All of these programming languages can, and is currently being, used for similar applications hence it is important to compare the differences in readability, writability and reliability of the languages, and the trade-offs that accompany the differences. The study consists of a theoretical comparison between the constructs and design of the programming languages to judge readability, writability and reliability, a survey conducted

XXX-X-XXXX-XXXX-X/XX/$XX.00 ©20XX IEEE

using code snippets to judge just readability and writability. Finally the readability and writability were compared to the runtimes of two algorithms to analyze whether there's a trade-off between efficiency and these two factors, and we also tried to check for reliability issues while running these computationally intensive algorithms.

II. Literature Review

For the purpose of this research we went with a more conventional assortment of algorithms and bits of code that are commonly employed. However, there are far more excruciating ways that accentuate the limits of what programming languages can output due to the way they were designed. In a study, Fourment et al. used standard bioinformatics programs- the Sellers algorithm, the Neighbor-Joining tree construction algorithm, and an algorithm for parsing Blast file outputs. They compared the memory usage and speed of execution implemented in 6 different programming languages. The results yielded C and C++ to be faster and use the least memory, however, Perl and Python were much more flexible compared to all the other languages. Whilst, C# appeared to be a bit of a compromise between the flexibility of Perl and Python and the efficient performance of C and C++ [3].

In another work, Bhattacharya took a different approach in benchmarking the programming languages C and C++. They took in factors like developer competence and the whole development process in general. They conducted studies on statistical analysis on a set of long-lived, widely-used, and open source projects like Firebox, Blender, VLC, and MySQL, which already have a substantial amount of portions of development in C and C++ [6]. They found out that C++ yielded more reliable and better quality outcomes.

In another literary article the researchers came up with an evaluation Framework and comparative analysis of the widely used first programming languages. Farooq et al. sought to glorify the importance of computer programming and its pivotal contribution to the computing curricula. The goal of the research was to find a way to evaluate the suitability of a programming language as an FPL (First programming language) [5].

A similar study conducted by David Gries at the Computer Science Department of Cornell University discussed what should be taught in an introductory programming course and tied the readability, writability, and reliability of different programming languages [7].

III. Readability, Writability and Reliability

Before embarking on the journey of comparing these three metrics for each language, it is important to understand how each is defined and what they signify.

A. Readability

As the term suggests, "readability" of a language is essentially how easy it is to read a piece of code and understand what it is doing [1]. When earlier programming languages were developed, the main factor to be considered was efficiency of the language. However, as the use of these languages increased, it became apparent that if users were taking a long time understanding an existing piece of code, then it was slowing down the process and hence reducing efficiency before the computation even began. Hence, eventually the focus shifted from machine oriented efficiency to human efficiency. Depending on the problem domain, the concept of readability changes. For example, a language that is not meant for complex numerical computations, such as vanilla JavaScript, might not make a very readable script for mathematical computations.

B. Writability

The writability of a programming language is defined by the ease of creating a new program for a specific problem domain [1]. Similar to readability, writability of a language is also heavily influenced by the problem domain it is being used in. An easy and common example for this would be the difference in writability for a program with a Graphical User Interface (GUI). A language such as Visual BASIC or Java would have high writability for such programs as they are designed for such applications, whereas C would have very low writability as it is simply not designed for programs that require a GUI.

C. Reliability

The performance of a program depends considerably on the language it has been written on. Its affects the decision of whether the program will be reliable for further extension and modification in the future if required. Of course reliability is a more abstract term and not all languages need to have the same standard and it depends on the purpose of the program it's written with.

A significant criterion to base our judgment has been mentioned in John D Gannon and J.J Horning's paper "Language Design for Programming Reliability"[8] which is the ability of a programming language to decrease programming errors and the ability to detect them if any. This can be related to the readability and writability of the said programming language. A language that is more readable theoretically should have less fuss to deal with when trying to detect anomalies or errors.

D. Factors and properties of languages to consider

TABLE I. Factors affecting Readability, Writability and Reliability

Readability	Writability	Reliability
Simplicity	Simplicity	Simplicity
Orthogonality	Orthogonality	Orthogonality
Data types	Data types	Data types
Syntax design	Syntax design	Syntax design
Comment style	Variable naming conventions	Support for abstraction
Indentation/White spacing	Support for abstraction	Expressivity
Variable naming conventions	Expressivity	Type checking
		Exception handling
		Restricted aliasing

In the table above, factors that affect readability, writability and reliability are shown. Short descriptions of each factor is discussed below.

1) Simplicity

Simplicity of a language is exactly what the term suggests, and there are many factors that affect simplicity of a language. Firstly, the complexity and size of the language constructs of a programming language affect its simplicity. Languages that have a large number of basic syntaxes are often difficult to

learn, and often they are not fully mastered by a programmer. This poses as a problem when a reader faces constructs they are not familiar with and finds the code unreadable, even though they might be an expert with a different subset of the language [1].

Another issue that reduces simplicity of a language is feature multiplicity. When the same task can be carried out in multiple ways using different syntax, the language is said to have high feature multiplicity; this makes the language more complex as someone who is habituated to using one, will have a hard time understanding the other. [1]

Operator overloading also increases the complexity of a language. While it is usually seen as an increase in flexibility, it also creates the possibility for users to create operations that do not align with the function's original meaning or is distinctly different from it. For example, a user might define a function subtract using the "+", and that is possible through operator overloading. Hence this will greatly confuse the reader or future contributor, immensely reducing the readability.[1]

2) Orthogonality

The orthogonality of a language can be defined in two ways. One, it is the concept of one operation doing only one task at a time and not affecting any other variable or condition [9], in other words, there is no side effect of the operation and there is only one way to do the operation. Another definition is that, all language constructs are independent of each other, hence they can be logically combined together to control and make data structures as necessary [1]. If the constructs are independent of each other, they will not behave differently in different contexts. Hence, languages that have context-dependent syntax are not orthogonal. Moreover, languages that have many exceptions in rules, such as return types or parameter passing, are not orthogonal.

3) Data types

There should sufficient numbers of data types present in the language such that all types and sizes of data can be accurately represented and stored in the memory [1]. Some languages do not have a primitive data type for string or Boolean, and this often reduces the readability and writability of the language as these variables have to be expressed as something they are not.

4) Syntax Design

Syntax is essentially the building blocks of a language. Logical syntax that reflect their purpose clearly is an obvious aid to readability. Another point to consider is the way compound statements are designed. In many languages, compound statements or statement groups are not easily readable as it is hard to judge the hierarchies of the statements, and where they began/end.

5) Comment style

Comments make a code readable as they usually describe what is happening in the code and its purpose. However, how these comments are structured affect reliability as well. There are mainly 2 types of comments: single line comment and multi-line comment/ block comment. Among programmers, it is general consensus that single line comments are more readable than block comment, as it often becomes difficult to understand where the comment stops and code starts again, especially if someone is using a non-color coded environment [5].

6) Indentation/Whitespace

White spacing, or gaps in between lines of code, are treated differently in different languages. In some languages, white spaces are simply disregarded and no indentation is needed for the code to function properly. However, indented code is far more readable than non-indented code, as the indentations make the hierarchy of compound statements much more apparent. Therefore, languages that enforce indentation for loops or conditional statements, have much better readability. Languages that enforce every new instruction to be on a new line are also more readable.

7) Variable naming conventions

Conventions enforced by a programming language can be seen as both increasing/decreasing the readability. This is highly subjective, as someone who is familiar with the conventions will find the code much more readable when the conventions are followed, whereas someone who is accustomed to another convention might find the same code strange. However, it is general consensus that having some set conventions is better as it encourages the programmer to follow better programming practices.

8) Support for abstraction

When it comes to readability and writability, support for abstraction is paramount. Using classes to encapsulate elements of an object, using functions to declare procedures only once and use them multiple times, all contribute to making the code readable and cluster-free..

9) Expressivity

Expressivity refers to the power of a language by which it can express complex procedures using short notation or fewer lines of code. This in turn leads to a bigger number of constructs in the code, some of which are quite powerful and hence might reduce the overall simplicity of the code. However, fewer lines of code usually reduce the time needed for a reader to understand it [1].

10) Type checking

All languages check for type errors now, but the main question is when this checking is done. Run-time type checking reduces the efficiency of the program greatly, hence compile-time type checking is more reliable and desirable. Moreover, the earlier errors are detected (usually by compiler), the easier and less expensive it is to fix the error.

11) Exception Handling

Regardless of the number of checks done in compilation, there will always be unforeseen circumstances in runtime. What facilities a language provides in dealing with these unpredictable or predictable unwanted circumstances is important, as otherwise programmers cannot make a stable program that runs reliably or gives reproducible results.

12) Restricted Aliasing

Aliasing is when two pointers are directing to the same memory location. This is undesirable as we don't want mismatched data, undefined results or overwrite any value. Thus a strict aliasing rule is often integrated within the language system to prevent pointers of different objects to never indicate to the same memory location.

In Table II, these metrics have been summarized together.

TABLE II. METRICS TO DEFINE READABILITY, WRITABILITY AND RELIABILITY OF LANGUAGES

Simplicity	**Orthogonality**	**Data type**	**Syntax design**	**Comment style**	**Whitespace /indentation**	**Variable naming conventions**	**Support of abstraction**	**Expressivity**	**Type checking**	**Exception handling**	**Restricted aliasing**
Number of constructs (lesser the better)	Context dependent syntax? (should not be)	Support available for all data necessity? (should be available)	Form and meaning (should be related to the meaning)	Allows single line? (should allow)	Indented? (should be)	Enforced by language? (should be)	Level of support (should be high level support)	Average number of lines needed to write a program (should be fewer)	Done in compile time or run-time? (should be compile time/ statically)	Available? (should be available)	Provided? (should be provided)
Feature multiplicity (should not be supported)	Return types (should be consistent)		Compound statements (should have definitive ways to signify hierarchy)	Allows multi line? (should not allow)	White spacing disregarded? (should not be)				Strongly typed? (should be)		
Operator overloading (should not be supported)	Exceptions in rules(return types, parameter passing) (should be consistent)										

IV. PROGRAMMING LANGUAGES

As mentioned earlier, the six programming languages used in this research are C, C++, Java, JavaScript, Python and R. In this section, some basic information about the languages are given, to provide some context about certain features they may or may not have.

A. C

C is one of the earliest and most impactful programming languages in history. It was developed in the early 1970s at the Bell Laboratory by American computer scientist Dennis M. Ritchie and Ken Thompson. It was initially developed to be used in UNIX operating system. When C was introduced it had a great combination of features as it was simple, portable, structured and extensible. It also had a richer library and better memory management system with pointer and recursion ability than other languages at that time.

B. C++

C++ was created by Bjarne Stroustrup as an extension of C programing language that introduced class functionality, which is why it was initially called C with classes. It was renamed C++ in 1983. The language started to expand its features with classes, basic inheritance, function argument, virtual functions etc.. Without compromising speed and portability, C++ has kept expanding its feature set to stay relevant decade after decade.

C. Java

A small team of engineers called Green Team in Sun microsystems initiated the JAVA language project in 1991. It was initially developed for interactive television like set top box, but was very advance for cable television back then. Java is robust, secured, platform independent, high performance object-oriented programming language and it is reasonably fast. Java provides a software based platform. Java virtual machine is what make java so portable.

D. JavaScript

JavaScript, was created by Brendan Eich at Netscape. JavaScript is one of the core technology of World Wide Web. It enables interactions in website. It is a multi-paradigm language that supports event-driven, functional programming style.

E. JavaScript

Guido van Rossum started working on Python in 1989 as a successor to ABC programming language. It was first released in 1991 although, version 1.0 of Python was released in 1994. Although it is not as fast as other popular programming languages such as C/C++ or Java, its main point of focus is ease of learning, alongside short and easy-to-code syntax which made it one of the most readable programming languages. It is now widely used in technical fields like machine learning, artificial intelligence and others.

V. METHODOLOGY

For the purpose of our paper, we defined a unique procedure to compare the six languages based on readability, writability and reliability. First of all, we created an evaluation metric of three standards : ‘Bad’ , ‘Moderate’, ‘Good’ to judge each of the languages for every criterion in table 1. This is done to form a theoretical comparison based on the characteristics of each language. However, since the evaluation metrics are quite abstract, and very subjective, we chose to conduct a survey to check whether our theoretical comparison is reflected in the preference and judgement of actual users. Hence, we detailed a set of questions in the form of a survey which is further discussed in section 7b. Finally, we ran experiments to compare the runtimes of algorithms written from scratch in these 6 languages, to check how the readability and writability of a language affects its efficiency, if at all. Our assumption was that languages with high readability and writability are closer to the English language i.e.: higher level, hence they should be slower than comparatively lower level/ less readable languages.

A. Theoretical comparison

To visually explain the findings of our secondary research, we produced the following tables that concisely mentions the performance of each of the languages.

Judgement criteria used:

BAD	MODERATE	GOOD

In Table III, the factors that affect readability have been discussed. In Table IV, the extra factors that affect writability

TABLE III. COMPARISON BETWEEN FACTORS THAT AFFECT READABILITY

	Simplicity	Orthogonality	Data types	Syntax design	Comment style	Whitespace/ indentation	Variable naming conventions
C	Moderate number of constructs	Some constructs are context dependent [5]	Multiple data types available just for integer type, but no primitive type for string	Form and meaning reflects purpose	Allows single line comment	Not indented	Enforced by language
	Supports feature multiplicity[5]	Return types not consistent [5]		Compound statements signified by curly braces	Allows multi line comment	Whitespace disregarded	
	Does not support operator overloading	Has exceptions in rules [5]					
C++	Large number of constructs	Some constructs are context dependent [5]	Sufficient number of data types available	Some syntax do not reflect their meaning (such cout, cin)	Allows single line comment	Not indented	Enforced by language
	Supports feature multiplicity [5]	Return types not consistent [5]		Compound statements signified by curly braces	Allows multi line comment	Whitespace disregarded	
	Supports operator overloading	Has exceptions in rules [5]					
Java	Large number of constructs	constructs are not context dependent [5]	No support for complex numbers	Form and meaning reflects purpose	Allows single line comment	Not indented	Enforced by language
	High feature multiplicity[1]	Return types consistent[5]		Compound statements signified by curly braces	Allows multi line comment	Whitespace disregarded	
	Supports operator overloading	No exceptions in rules[5]					
JavaScript	Large number of constructs	Commonly used constructs are not context dependent	No support for complex numbers	Some syntax are a bit vague	Allows single line comment	Not indented	Enforced by language
	High feature multiplicity	Return types consistent		Compound statements signified by curly braces	Allows multi line comment	Whitespace disregarded	
	Does not support operator overloading	Few exceptions in rules					
Python	Fewer number of basic constructs	constructs are not context dependent [5]	Everything is an object, proper support available for data types	Form and meaning reflects purpose, except list and tuples	Allows single line comment	indented	Enforced by language
	Mostly no feature multiplicity [5]	Return types consistent[5]		Compound statements signified by indented blocks	Does not allow multi line comment	Whitespace not disregarded	
	Supports operator overloading	No exceptions in rules[5]					
R	Fewer number of basic constructs	constructs are not context dependent	Everything is an object, proper support available for data types	Form and meaning reflects purpose, but some are quite different than common languages	Allows single line comment	indented	Enforced by language
	No feature multiplicity	Return types consistent		Compound statements signified by indented blocks	Does not allow multi line comment	Whitespace not disregarded	
	Supports operator overloading	No exceptions in rules					

have been discussed and finally in Table V, the remaining factors that affect reliability only have been discussed.

Based on our findings so far, Python and R seem to be the most readable and writable. Both of these languages have become very prominent languages in recent years, which might be attributed to their high level readability and writability. C has very low readability and writability, which makes sense considering it is the oldest and lowest level language amongst these 6 languages.

TABLE IV. COMPARISON BETWEEN FACTORS THAT AFFECT WRITABILITY

	Support for abstraction	Expressivity
C	No support	About 5 lines needed to print hello world
C++	High level abstraction	About 6 lines needed to print hello world
Java	High level abstraction	About 5 lines needed to print hello world
Python	High level abstraction	1 line to print hello world
JavaScript	No built in support	1 line to print hello world
R	High level abstraction	1 line to print hello world

However, in terms of reliability, Java seems to be the reigning champion. Java has been continuously developed for many years, and is one of the main languages used in industry level development and software. Hence it does make sense that it has the most reliable characteristics. JavaScript and R both have low reliability, which is consistent with the fact that JavaScript is mostly used in web and server applications, but not other types of software and R is not used for software development at all.

B. *Survey*

The main purpose of this survey was to check whether people's opinions reflected the findings found in the theoretical comparison. However, we also wanted to see if there was a difference in preference or judgement of readability and writability between non/new programmers and experienced programmers. Our assumption is that there should be some differences, as programmers tend to have a bias for the languages they're more familiar with and find them or languages with similar syntax more readable. The questions asked the participants the following questions: how familiar they were with programming, how important readability and writability was to them, and then a code snippet of a bubble sort algorithm written in each language was shown and they were asked to score the languages based on readability and writability, in separate questions.

C. *Efficiency comparison*

It is general knowledge that high level languages are comparatively slower than lower level languages, due to the high amounts of abstraction and layers that have to be read and decoded before the code can be converted into machine language. Higher level languages have been designed to be more readable than lower level languages, hence we wanted to check if there was a difference in efficiency of a language and its perceived readability/writability. For this purpose, two algorithms were used: Bubble sort algorithm and Matrix multiplication algorithm. To keep the factor of abstraction somewhat constant between the languages, available library functions were not used for sorting or multiplication, instead the algorithms was implemented from scratch. The only exception was R in matrix multiplication, which has a designated operator specifically for matrix multiplication. The code snippets for the bubble sort algorithm have been included as code listings, but for the sake of space, matrix multiplication codes have been submitted separately. The same codes, but in a more readable format, were used in the survey as well.

TABLE V. COMPARISON BETWEEN FACTORS THAT AFFECT RELIABILITY

	Type checking	Exception handling	Restricted aliasing
C	Static type checking Not very strongly typed[5]	Does not provide support	Support provided
C++	Static type checking Mostly strongly typed[5]	Moderate support [5]	Support provided
Java	Static type checking Mostly strongly typed[5]	Strong support using catch and throws	Handles aliasing in runtime, so moderate support
JavaScript	Dynamic type checking Not strongly typed	Moderate support using try catch throw	Aliasing not possible
Python	Dynamic type checking Mostly strongly typed[5]	Strong support using try except[5]	Support not provided
R	Dynamic type checking Strongly typed	Moderate support	Aliasing not possible

Code listing 1: Bubble sort in C

```
#include <stdio.h>
void swap(long long int *xp, long
long int *yp)
{
   long long int temp = *xp;
   *xp = *yp;
   *yp = temp;
}
void bubbleSort(int arr[], int n)
{
  int i, j;
  for (i = 0; i < n-1; i++)
     for (j = 0; j < n-i-1; j++)
        if (arr[j] > arr[j+1])
           swap(&arr[j], &arr[j+1]);
}
int main(void)
{
   int arr[1000000];
   int i = 0;
   FILE * fp;
   if (fp = fopen("numbers.txt",
"r")) {
      while (fscanf(fp, "%d",
&arr[i]) != EOF) {
         ++i;
      }
      fclose(fp);
   }
   int n = sizeof(arr)/sizeof(arr[0]);
   bubbleSort(arr, n);
   printf("Sorted array: \n");
   for (i=0; i < size; i++)
      printf("%d ", arr[i]);
   printf("\n");
   return 0;
}
```

Code listing 2: Bubble sort in C++

```
#include <fstream>
#include<iostream>
using namespace std;
void swap(int *xp, int *yp)
{
   int temp = *xp;
   *xp = *yp;
   *yp = temp;
}
void bubbleSort(int arr[], int n)
{
   int i, j;
   for (i = 0; i < n-1; i++)
   for (j = 0; j < n-i-1; j++)
      if (arr[j] > arr[j+1])
         swap(&arr[j], &arr[j+1]);
}
int main() {
  int arr[100000];
  ifstream is("numbers.txt");
  int cnt= 0;
  int x;
  while (cnt < arr[100000] && is >> x)
  arr[cnt++] = x;
  int n = sizeof(arr)/sizeof(arr[0]);
   bubbleSort(arr, n);
   cout<<"Sorted array: \n";
   for (i = 0; i < size; i++)
      cout << arr[i] << " ";
   cout << endl;
  is.close();
}
```

Code listing 3: Bubble sort in Python

```
def bubblesort(arr):
   n = len(arr)
   for i in range(n):
      for j in range(0, n-i-1):
         if arr[j] > arr[j+1] :
            arr[j], arr[j+1] = arr[j+1],
arr[j]
arr = list()
filename = 'numbers.txt'
with open(filename) as fin:
   for line in fin:
      arr.append(line)
print(arr)
bubblesort(arr)
print ("Sorted array is:")
for i in range(len(arr)):
   print ("%s" %arr[i])
```

Code listing 4: Bubble sort in R

```
bubblesort <- function(arr){
  n = length(arr)
  v = arr
  for(j in 1:(n-1)){
    for(i in 1:(n-j)){
      if(v[i+1]<v[i]){
        t = v[i+1]
        v[i+1] = v[i]
        v[i] = t
      }
    }
  }
  arr = v
}
y<-scan(file = "numbers.txt", what =
numeric(), sep = "\n")
sortedarray<-bubblesort(y)
print("Sorted array is:")
sortedarray
```

Code listing 5: Bubble sort in Java

```
package BubbleSort;
import java.io.File;
import
java.io.FileNotFoundException;
import java.util.Scanner;
public class Main {
   static void bubbleSort(int[] arr) {
      int n = arr.length;
      int temp = 0;
      for (int i = 0; i < n; i++) {
         for (int j = 1; j < (n - i); j++)
{
            if (arr[j - 1] > arr[j]) {
               //swap elements
               temp = arr[j - 1];
               arr[j - 1] = arr[j];
               arr[j] = temp;
            }
         }
      }
   }
   public static void main(String[]
args) {
      int[] tall = new int[1000000];
      try {
         Scanner scanner = new
Scanner(new
File("E:\\Codes\\cse425_benchmar
king\\numbers.txt"));
         int i = 0;
         while
(scanner.hasNextInt()) {
            tall[i++] =
scanner.nextInt();
         }
      } catch
(FileNotFoundException
filenotfoundexception) {
         System.out.println("File not
found.");
      }
      bubbleSort(tall);
      for(int i=0;i<tall.length;i++)
//length is the property of the array
         System.out.println(tall[i]);
   }
}
```

Code listing 6: Bubble sort in JavaScript

```
function bubble_Sort(a)
{
   var swapp;
   var n = a.length-1;
   var x=a;
   do {
      swapp = false;
      for (var i=0; i < n; i++)
      {
         if (x[i] < x[i+1])
         {
            var temp = x[i];
            x[i] = x[i+1];
            x[i+1] = temp;
            swapp = true;
         }
      }
      n--;
   } while (swapp);
 return x;
}
var fs = require('fs');
var text =
fs.readFileSync("./numbers.txt").toString('
utf-8');
var textByLine = text.split("\n")
console.log(bubble_Sort(a));
```

VI. Results and Analysis

In this section, the results of the survey and efficiency experiments have been discussed. The results of each section is followed by its analysis, however farther research is needed to strongly establish our reasoning.

A. Survey results

A total of 58 participants filled out the survey which was carried out using Google forms. Rather than targeting programmers only, non-programmers were asked for their opinions as well. One important point to note is that the survey was done over a short duration of 4 days only, and most of the participants were students of the same university. Hence, farther more rigorous surveys or methodologies are needed to strongly claim any of the findings discussed in this section. A brief summary of the results is given in Table VI below.

TABLE VI. SUMMARY OF SURVEY RESULTS, MAINLY IMPORTANCE OF READABILITY AND WRITABILITY

Metrics	All participants	New/Non-programmers	Experienced programmers
Number of participants	58	11	47
Mean of importance of readability score	3.62069	3.454545	3.659574
Mean of importance of writability score	3.706897	3.363636	3.787234

In Table VI, it can be seen that the number of experienced programmers is much higher than non-programmers. This is a limitation of our survey as it was done in a short span of time, hence it was not possible to distribute it to a larger, non-biased demographic. It is a logical deduction to make that the general results will be skewered towards the experienced programmers' opinions. Another finding we noticed is that the importance of readability and writability is slightly higher to experienced programmers, which can be attributed to the logic that non/new programmers are unaware about the consequences of poor readability and writability.

In Fig.1, the boxplots of each language for the readability scores are shown for each category.

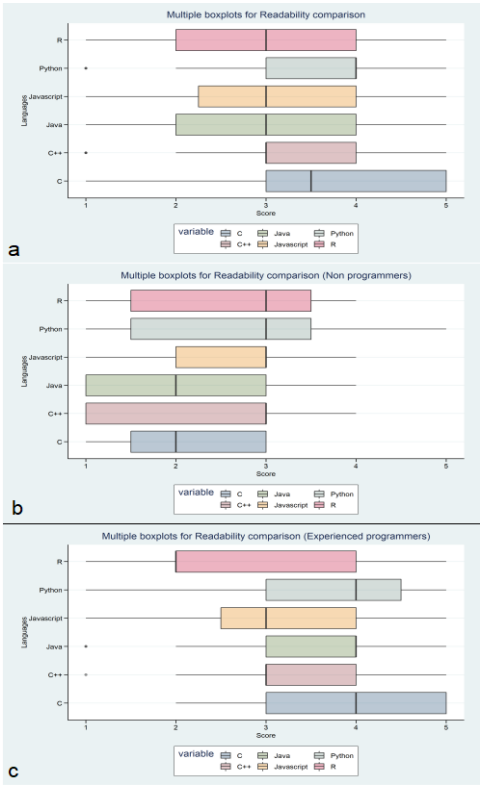

Fig. 1. multiple boxplots showing the readability scores of each language for a) all participants, b) non/new users, c)old users

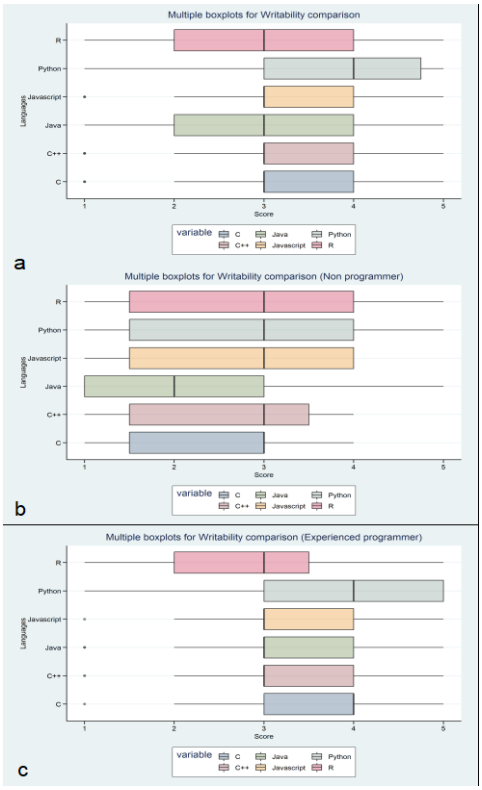

Fig. 2. multiple boxplots showing the writability scores of each language for a) all participants, b)non/new users, c) old users

From Fig. 1, the first finding is that majority of the participants found C to be most readable as the third quartile is much bigger than the first quartile and stretches over score 4 and 5, and this is explained by the preference of experienced programmers as well. This is a possible bias in the dataset as most of the participant of this survey are students majoring in Computer Science in North South University, where the first programming language taught is C. The bias is farther proved as the same is not seen from the perspective of new programmers, where both C and C++ have lower readability scores. Scores from 1 to 3 seem to be more prominent in the new programmers' boxplots, which can be attributed to the fact that non-programmers will find most code less readable due to their lack of experience. However, this inexperience also signifies the lack of bias, as they are judging purely based on the visual characteristics. To summarize all 3 boxplots, new programmers seem to find Python and R more readable, which is consistent with our theoretical findings. Experienced programmers are geared more towards C and Python, but this cannot be solidly claimed due to the possible bias.

In Fig.2, the boxplots of each language for the writability scores are shown for each category.

Judging from the boxplots showing writability in Fig.2 , most languages seem to have similar distributions except R and Java, which show lower range of scores. Non/new programmers score Java the lowest, which might be because Java's I/O syntax is most different amongst all the languages. Other than that, no particular trend can be seen as the scores are across a wide range of values. Experienced programmers show strong liking towards Python and find R to least writable. Java, JavaScript, C++ and C have very similar distributions.

In Fig.3, we have plotted the correlation between the scores of each language and compared the matrices between the new/non-programmer and experienced programmer categories. It can be seen that C and C++ are quite highly correlated, which is what we expected as C++ has been developed based on C. However, the correlation is not as strongly seen in experienced users, which farther confirms our deduction that there was bias in their opinions. It is safe to assume that programmers who have worked with C but not C++, or vice versa, do not find the two languages similar in terms of readability. New programmers also find R to be quite similar to Python and JavaScript, which is probably due to the fact that the code snippets of these languages had similar lengths and some syntax as well. No significant correlation can be seen in the scores obtained from experienced programmers.

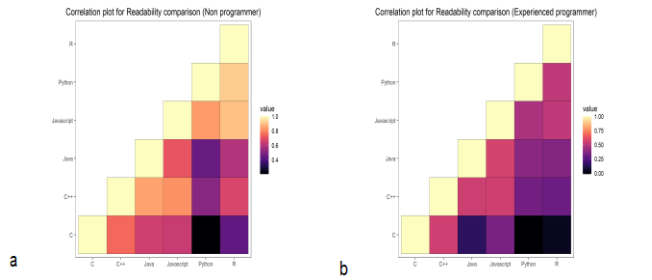

Fig. 3. Correlation plots comparing the readability of the languages between a)new users and b)old users

B. *Efficiency comparison results*

As mentioned before, two algorithms, bubble sorting and matrix multiplication, were coded in each language for the experimentation. The pseudocode of each program was kept

as consistent as possible, to avoid timing differences due to difference in steps. Each experiment was repeated three times to reduce error in timing calculations. All the experiments were run on the same hardware to avoid hardware dependencies, and the specifications were: Core i7-7700HQ processor and 16 GB RAM. Our threshold time was 2 hours due to shortage of time, after which we manually stopped the execution.

Bubble sort was first applied on 100,000 random numbers stored in a text file and the same process was repeated for 1000,000 numbers. The results are given below in Table VII.

TABLE VII. RUNTIMES IN SECONDS FOR EACH BUBBLE SORT EXPERIMENT

	C	**C++**	**Java**	JavaS**cript**	**Python**	R
100,000 numbers	43.916	56.135	1340	>2 hours	1190.527	922.1478
1000,000 numbers	Garbage values	51.40133333	1769.533333	>2 hours	>2 hours-	>2 hours-

While running these experiments some reliability issues were apparent. For one, C could not handle the calculations when there were 1000000 numbers and just generated garbage values. This could be due to memory management, however we did not try to manually manage the memory as: 1) it would affect the readability and writability of the code, and 2) it would change the overall pseudocode followed. JavaScript was unable to finish both the experiments in 2 hours, whereas Python and R could not finish when 1000000 numbers used.

The matrix multiplication algorithm used was suitable for square matrices, hence a square matrix of 100 x 100 was made by reading 10000 numbers from a text file and storing it in 2 separate matrices to multiply them. The results for this experiment are given below in Table VIII.

TABLE VIII. RUNTIMES IN SECONDS FOR MATRIX MULTIPLICATION EXPERIMENT

	C	**C++**	**Java**	**R**	**Python**	**JavaScript**
100x 100 matrix	1.1106	2.4626	1.6667	1.7667	>2 hours	>2 hours-

Reliability issues were seen while running the matrix multiplication experiment as well. First of all, the use of file pointers in C and C++ is quite complicated, as the pointer moves with every input that is read and stays on that point. To read a file from the beginning again, the pointer needs to be manually relocated to the beginning of the file again. This is a reliability issue as most languages do not require this to be done manually, such as Python, R or even Java. Not resetting the pointer leads to garbage values.

Regardless of the reliability issues, from the runtimes it can be seen that C is clearly the fasted IF it can handle the operation. On the other hand, Python and JavaScript had much slower runtimes in both algorithms. Relating these findings to the previous findings that C is one of the least readable language (both theoretically and from the survey), and Python is one of the most readable language (both theoretically and from the survey), we would like to claim that greater readability does have a negative impact on efficiency of a programming language. Judging from the evidence so far, this is mostly due to the high level of abstraction and consequent distance of the code from the compiler level. Also factors such as dynamic binding make runtimes more expensive as type checking and bindings of variables are done in runtime rather than compile time.

VII. DISCUSSION

While this research was intended to compare six languages based on their readability, writability and reliability, it is in no way a complete study. It is important to define a mathematically sound methodology to properly assess these metrics, as they are quite subjective. Issues that were faced during this research included lack of resources, as not much work has been done on this topic. Hence, this is a field with potential that should be explored. We also faced hardware issues as much of the calculation is RAM dependent, which is why repeating the computationally intensive algorithms for bigger numbers was not possible. Finally, the scope of the survey was not wide enough to make a sold claim.

VIII. CONCLUSION

In this research, six popular, mainstream languages have been analyzed based on their readability, writability and reliability. According to our theoretical findings, Python and R were the most readable and writable, whereas Java has an advantage on the factors that only affected reliability. In our survey, not much could be concluded from the general results, however dividing the results into the non/new-programmer and experienced programmers categories showed some interesting information. It could be seen that, new/non-programmers chose Python and R as the more readable languages, while ranking Java, C++ and C as the least readable language, which supports our theoretical findings. We also found that there is a possible bias in experienced programmers to the language they're most familiar or comfortable with, which is predictable yet interesting. Hence, based on readability and writability, Python is the winner in this research.

However, based on reliability, the only language in which we did not face any issue was Java. Java was able to complete all the experiments smoothly, and had no unpredicted issues that hampered our coding. This supports out theoretical findings well, hence according to this research, Java is the most reliable language amongst this six programming languages.

ACKNOWLEDGMENT *(Heading 5)*

We would like to thank our faculty supervisor, Md. Shahriar Karim, for his constant support and prompt feedback, and also for giving us the opportunity of doing this research.

REFERENCES

[1] R. Sebesta, Concepts of programming languages.

[2] R. Ihaka and R. Gentleman, "R: A Language for Data Analysis and Graphics," J. Comput. Graph. Stat., vol. 5, no. 3, pp. 299–314, 1996, doi: 10.1080/10618600.1996.10474713.

[3] M. Fourment and M. R. Gillings, "A comparison of common programming languages used in bioinformatics," BMC Bioinformatics, vol. 9, pp. 1–9, 2008, doi: 10.1186/1471-2105-9-82.

[4] G. van Rossum, "Python tutorial, May 1995," CWI Rep. CS-R9526, no. CS-R9526, pp. 1–65, 1995, [Online]. Available: http://oai.cwi.nl/oai/asset/5007/05007D.pdf.

[5] M. S. Farooq, S. A. Khan, F. Ahmad, S. Islam, and A. Abid, “An evaluation framework and comparative analysis of the widely used first programming languages,” PLoS One, vol. 9, no. 2, pp. 1–25, 2014, doi: 10.1371/journal.pone.0088941.

[6] P. Bhattacharya, “Assessing Programming Language Impact on Development and Maintenance : A Study on C and C ++ Categories and Subject Descriptors,” Methodology, no. 2, pp. 171–180.

[7] D. Gries, “What should we teach in an introductory programming course?,” Proc. 4th SIGCSE Tech. Symp. Comput. Sci. Educ. SIGCSE 1974, pp. 81–89, 1974, doi: 10.1145/800183.810447.

[8] J. D. Gannon and J. J. Horning, “Language Design for Programming Reliability,” IEEE Trans. Softw. Eng., vol. SE-1, no. 2, pp. 179–191, 1975, doi: 10.1109/TSE.1975.6312838.

[9] E. S. Raymond, “UNIX Philosophy,” art UNIX Program., p. 560, 2003, [Online]. Available: http://portal.acm.org/citation.cfm?id=829549.