J. Basic. Appl. Sci. Res., 2(8)8144-8149, 2012

ISSN 2090-4304
Journal of Basic and Applied Scientific Research
www.textroad.com

A Framework for the Assessment of First Programming Language

Muhammad Shoaib Farooq[1], Sher Afzal Khan[2] and Adnan Abid[1,3]

[1]Faculty of Information Technology, University of Central Punjab Lahore, Pakistan
[2]Department of Computer Sciences, Abdul Wali Khan University, Mardan, Pakistan
[3]Department of Electronics and Information, Politecnico di Milano, Milan, Italy

ABSTRACT

Choosing an appropriate programming language to be taught as the first programming course in the computer science and computer engineering disciplines has been a debatable issue. Many languages have been practiced for this purpose, namely, Fortran, Pascal, Modula-2, Ada, C, C++ and Java. However, in terms of teaching, each language has its own pitfalls and shortcomings as first programming language. For example, some languages are successor of another language and adapt legacy which results into ambiguous features, some are not widely accepted in the industry. Furthermore, academic institutions are forced to use these languages in syllabus due to industry relevance. A big challenge is that no criteria or de-facto standard exists for the assessment of first programming language. Therefore, in this paper we propose a framework for the suitability analysis of first programming language for the software engineering discipline. This framework will help in assessing the suitability of any programming language as first programming language in the computer science and computer engineering disciplines.
KEYWORDS: First Programming Language, Object Oriented Programming Language, Software Engineering Discipline.

1. INTRODUCTION

Programming holds a central position in the core computer science curriculum [14]. The programming language chosen to teach the first course in computer programming is called the First Programming Language (FPL).The major objective of teaching FPL is to build conceptual understanding of the programming constructs, and impart the knowledge to the students so as to build the abilities of problem decomposition and specification [11]. A better understanding of the concepts of language constructs and various parts of the programming process is significant and helps the students improve their skills to continue with advanced computer programming.

The first programming language has a great significance as it lays the foundation of the student for the subsequent courses in the discipline [2][13]. Generally, the students consider that the objective of the FPL course is to master the syntax of the language. Therefore, it is imperative to guide and educate the students with sufficiently sound theoretical foundation along with rigorous implementation exercise. This will put the students in a better position to design and implement the required software. An important consideration for the FPL is that it should be easy to learn with a small learning curve, so that the students can balance their learning by improving problem solving skills [5].

In this paper, we have presented a framework for the selection of an appropriate First Programming Language for computer science and computer engineering discipline. This framework comprises of several functional and non-functional criterion so as to choose an appropriate language. We argue that our proposed framework will help selecting an appropriate FPL.

2. RELATED WORK

Currently, there are various different languages that are being taught as a first programming language in the educational institutes, at different levels, all over the world. These languages are from Pascal to Scheme and from C++ to Java and from Modula-2 to JavaScript. Yet there is no defacto standard language that exists as FPL. The reason is that it is the discretion of the institution or the faculty to choose a particular language to be taught as FPL. Therefore, the selection of an appropriate programming language for an introductory course in computer programming requires serious attention, as Bjarne Stroustrup states, “the choice of a first language is always controversial” [12].

***Corresponding Author**: Muhammad Shoaib Farooq, Faculty of Information Technology, University of Central Punjab Lahore, Pakistan

Table 1: Percentage of leading first programming languages taught in the world

Languages	Percentage
Ada	8
C	6
C++	33
Fortran	2
Java	8
Modula-2	7
Pascal	18
Scheme	9
Other	9

Out of the many contemporary programming languages being taught C++ is the most widely used as FPL in the major European and American universities. Table 1 shows the percentage of the leading first programming languages taught in the world [7].

3. FRAMEWORK

Below we present a framework which comprises of two types of characteristics. Firstly, we mention some technical characteristics that the language should hold. Secondly, we present the other set which is composed of technical features. We argue that a healthy FPL should adhere to our specified features, and together, both types of characteristics provide a set of comprehensive assessment criterion for the selection of an appropriate FPL. Firstly, we discuss the technical characteristics, and later on, we present the set of non-functional features in our proposed framework.

3.1 TECHNICAL FEATURES

3.1.1 FEATURE UNIFORMITY. A language holds feature uniformity if there is no proper subset of that language that can solve all problems which can be solved by whole set. The language should be as small as possible, while including all features that are required for the first programming language. In case of a larger language which contains redundant features, the instructor has to teach a subset of the language. A subset is good for the purpose of writing the code however, for reading one must know everything, which is not desirable. In most cases smaller languages are easy to learn and provide a stronger base to the beginners to develop their problem solving skills. It also produces a smaller learning curve.
It is hard to find any such language which holds the feature uniformity. As an example, C++ performs the swapping operation in two different ways. Many languages hold redundant features because of the backward compatibility, which also leads to an increase in the language complexity, in terms of the number of constructs.

3.1.2 HIGH LEVEL. A beginner in computer programming should not be forced to deal with the machine internals. Tasks that can easily be carried out by the compiler, or by the runtime system, should not be the responsibility of a programmer. IBM defines the level of a language as the number of assembly language statements it will take to produce the functionality of one statement in the target language.
In a high level language one instruction should be equal to three or more assembly language instructions [8]. Table 2 shows levels of popular leading first programming languages. Except C all languages are high level as shown in Table 3.

Table 2: Levels of leading First Programming Languages

Languages	Level
Ada	6.5
C	2.5
C++	6
Fortran	4
Java	6
Modula-2	4
Pascal	4
Scheme	6

An example of violation of this requirement is the explicit management of dynamic storage. Putting the responsibility for storage management into the hand of beginner programmer is unnecessary and may lead to frustrating experiences.

3.1.3 ORTHOGONALITY. A language is considered orthogonal if: [4]

1. A symbol or a reserved word always carries the same meaning regardless of the context in which it is used. All keywords are reserved.
2. For everything we want to do in the language, there should be one and only one way of doing it. There should be no feature multiplicity.
3. The language contains small number of basic features that interact in a predictable fashion no matter how they are combined in a program. The meaning of a construct should be combination dependent.

The more orthogonal the design is the fewer exceptions are there and it will make it easier to learn, read, and write. Too much orthogonality also cerates problem [4]. For Example in C – all statements return some value and hence can be used in expressions, e.g. assignment operator can be used as a boolean expression. But logic and arithmetic expressions can be intermixed. It will create readbility problem and can also cerate side effects. So there should be a midde ground.

3.1.4 STRONGLY TYPED. An important requirement is that the language should be strongly typed [2][6]. All type checking issues should be resolved at compile time. If any unexpected results occur they must not be the fault of the language rather fault of the programmer. So to make sure there are no unexpected results, types must be statically type checked and no automatic conversions should occur. The only possible way should be through explicit type casting.

C and C++ is strongly typed language except unions.Ada supports cleanly designed language constructs that are strongly typed.Type checking of unions requires that each union should include a type indicator, which is called a discriminant;this issupported by Ada and Pascal [4]. Fortran, C, and C++ provide union constructs in which there is no language support for type checking; the union in these languages is called free union [4]. Java and Scheme do not support unions.

3.1.5 CONSISTENT AND CLEAR. Ideally, the language and its environment should not allow side effects and dirty programs [2]. A consistent and clear language should not allow:

1. Coercion with demotion
2. Scope overriding
3. Unconventional Operator overloading

Implicit type conversion by compiler are called Coersion. There are two types of coesrion, promotion and demotion. In promotion only small size same family types can be assigned to same family large types. E.g. In Java short can be coerced to integer and in turn integer can be coerced to double (Code Listing 1).

Code Listing 1: Coercion with prmotion in java

```
class Test {
public staic void main (String a[]){
short a=5;
int b=a;//short coerced to integert
double c=b;// integer coerced to double
}}
```

C++ allows promotion as wells as demotion in coercion. Demotion may create data loss problem (Code Listing 2).

Code Listing 2: Coercion with demotion in C++

```
float a=2.5,b=3.2;
short c=a+b;
cout<<c; //the output is 5
```

Scope overriding affects the readability of the programs. Most of the block scope languages allow declaring variables with same names, although, the variable name should be unique within single block, yet nested blocks can have same name variable as parent block. C++ provides scope resolution operator for such type of variables. But, in some conditions may result into undesired results (Code Listing 3).

Code Listing 3: Infinite Loop due to Scope Overriding in C++

```
#include <iostream.h>
void main(){
int sameVariable=1;
while (sameVariable<10){
int sameVariable=1;
  cout<<" Loop forever"<<endl;
sameVariable++; //local variable in loop block
}}
```

Therefore, such issues can be seamlessly resolved by not allowing scope overriding of the variables, and the variables should be given unique names.
Some issues may also rise due to the operator overloading. As an example, in most languages the Division Operator '/' involves integer as well as real numbers as operands (Code Listing 4). Here, the result of the division process is truncated due to the data types of the operands, which results into mathematically wrong answer.

Code Listing 4: Division Operator in java violates the concept of basic mathematics

```
class DivisionProblem {
public  static void main(String o[]){
int dividend=5;
int divider=2;
   double result=dividend/divider;
   System.out.println(result); // it will print 2.0
   }}}
```

3.1.6 SECURITY. Low level progamming languages access the system resources and make it prone to security issues, as it results into uncontroled aliasing which is a major security threat. This may result into dangling references and garbage problem. Furthermore, for the beginners aliasing or creating references may cause issues. Also, the references can be brutal to the security of the program [10]. Keyword *new* without a corresponding *delete* will create problems while doing dynamic memoryallocation. The best way is to provide only the *new* keyword to the programmar, and hide d*elete*keyword from programmar. Deletion should be the responsibility of garbage collector.
For example, we can see a code in C++ which results into dangling references(Code Listing 5) and memory leakage (Code Listing 6).

Code Listing 5: Dangling Refernce Problem in C++

```
int *p1= new int;
*p1= 6;
int *p2=&p1
delete p1;
cout<< p2;//now  p2 has dangling reference
```

Code Listing 6: Memory Leakage Problem in C++

```
int *p1= new int;
p1= new int; // witout deleting create int type location
p1=new int;
```

Pascal also suffers from memmory leakage and dangling reference problem due to *delete* keyword in hand of programmar. In java, *new*is allowed but there is no *delete* in java. Garbage collector will delete all non refernced memory locations.

3.2 NON-FUNCTIONAL FEATURES. Now we present some non-functional features which we think are necessary for the framework which assesses the quality of the first programming language. The idea is to assess the language over important features which are not technical but are very relevant for learning a programming language in many other ways.

3.2.1 **Industry Relevance**. Most of the time a language is chosen to be taught based on the fact that how readily it is used in the software development in the industry.Many experts also have the same opinion that it is important to choose an "industrially strong" programming language for the early programming courses. Here, industrial strength refers to a languagethat is genuinely capable of being used for programming a software for the industrial and commercial purposes. There are several ways in which such a criterion may be met. A number of programming languages are popular in the educational institutes because of there significance in the software industry [2]. Some language are not so popular due to lack of modern features and less reuability.

C has limited usage in industry because it does not support primitive software engineeringprinciples and lacks modern features. Ada has a good share in the software Industry. C++ is quite popular because its was specially designed for professional programmers. Java is so popular in the industry due its civilized syntax and strong support of API(Application Programmar interface. Java is a mature language and currently one of the dominant languages in the software industry.

The main criteria for assessing the industrial relevance of a language is to survey the industry and see which language is widely being used for the implementation of medium to large scale projects.

3.2.2 **Less Effort for Writing Simple Programs**. The first programming language should require less programming effort to write the code of a program. This helps in improving the iterative learning process for beginners. It depends upon two things: firstly, on how many lines are required to write a simple program, and secondly,what level of learning overhead is required.

As an example, let us see the number of lines and learning overhead required to write a simple "Hello World!" program in different languages. Java requires the understanding of concepts like static, public void, and class to understand a simple "Hello Wolrd!" program (Code Listing 7).These are relatively advance features and are very difficult to learn at the beginning. Therefore, Java programmers have to learn many basic constructs of the language for writing simple program. Whereas, in C ++ the learning overahead requires the understanding of preprocessor #include, library header file, void, main, and program blcok(Code Listing 8).

Code Listing 7: Hello World Program in java

```
class HelloWorldExample{
public static void main (String a[]){
System.out.println("Hello World!!!");}}
```

Code Listing 8: Hello World Program in C++

```
#include <iostream.h>
void main (){
  cout<<"Hello World!!!";}
```

3.2.3 CONTEMPORARY FEATURES. Modern programming features and methodologies are always appealing for both academia and industry. So the first programming language should contain the modern features that are based on Software Engineering principles. These features include object oriented programming methodology, packages, generic programming and exception handling.

The closest to the real world applications and easy in understanding is the object-oriented paradigm. This paradigm is popular due to naturalness, reusability and easy implementation of problem domain. Apart from object orientation, effective exception handling and modular implementation are other main contemporary features. Exception handling is directly related to reliability and helps avoiding unusual events occurred at run time. Also, delivering the software to the customer with unhandled exceptions is highly unethical in Software Engineering. A good language should support proper exception handling mechanism.

Packages help obtaining modular software and also provide a way of organizing Application Programmer Interfaces. This packaging in turn helps in segregating the software into smaller pieces of code, where each piece is composed of closely related work. This helps in assembling different pieces together to develop a large program without the inherent complexity that comes with a larger program. By having a good modular design we can minimize coupling and maximize cohesion.

3.2.4 EASY TRANSITION. The concepts learned from the first programming language must conform to the theoratical concepts in the software design and development. This will help in mapping those concepts to language features and will help the students learning the art of programming instead of just learning a programming language. Hence, later on they can use all other languages which involve similar concepts.

3.2.5 READABILITY. Writing an easily readable code is also one of the desirable characteristic of our proposed framework for the assessment of a healthy FPL. Practically, this feature helps in many different ways. First of all, a readable program is more likely to be correct. While readable syntax, of course, does not guarantee correctness. Especially, in large scale systems where many people write the code, there are several instances where errors are discovered in programs only because a programmer did not understand another part of the code [1]. For both beginner and experienced programmers readability has clear advantages. It makes the learning of the language easier, helps to reduce the number of errors and makes the code easier to maintain [2][9].

Following are major issues in readable syntax: [4]

1. Identifiers Name should not be length dependent and there is no implicit declaration. E.g. in Fortran 77 an identifier can have 6 characters at most. Implicit declaration is also allowed in Fortran.
2. There should be consistent compound statement syntax. Special keywords for signaling the start and end of compound statement as shown in code listing 9.

Code Listing 9: end-if is used for termination

```
if (some condition) then
      do this
end-if
now do this
```

There should be proper block before else in order to avoid dangling else problem as shown in code listing 9.

Code Listing 10: Remedy for dangling else problem

```
if (some condition) then
    begin
       do this
    end
else
       do this
end-if
       now do this
```

3.2.6 USER-FRIENDLY ENVIRONMENT. It is desireable for the FPL to have easy to use and strongly integrated development environment (IDE). This IDEshould hide details of the underlying operating system and should allow students to focus on the programming task at hand. A good IDE should include features likesophisticatedlystructured editor, pretty printer, static checker and debugger. If we rate them in importance, the programming environment alone would probably be as important as the programming language itself [3]. Currently, user-friendly programming environment for all the major languages are easily available. All major first programming languages suport user friendly programming enviroment.

4. CONCLUSION

In this paper, we have presented a framework to assess the suitability of the first programming language for computer science and computer engineering disciplines. This framework is composed of two sets of characteristics; first set is composed of technical features, and the other is constructed from non-functional features. We have argued that our proposed framework is generic and systematic and will be a great help in the assessment of a suitable first programming language.

As a future work, we plan to assess the widely used first programming languages over our defined framework to assess their suitability as first programming language. This will not only enable us to find the most appropriate first programming language but will also highlight the deficiencies in these languages that can be addressed so as to make them a healthy first programming language.

REFERENCES

[1]. C.A.R. Hoare: The Emperor's Old Clothes, CACM, 24, 2, 1981.

[2]. M. Kölling: The Problem of teaching Object Oriented Programming Part1: Languages, Journal of Object Oriented Programming, 11(8): 8-15, 1999.

[3]. M. Kölling: The Problem of teaching Object Oriented Programming Part 2: Environments, Journal of Object OrientedProgramming, 11(9): 6-12, 1999.

[4]. R. Sebesta: Concepts of Programming Languages, Ninth Edition, Addison-Wesley, 2009

[5]. N. Wirth: On Design of the programming language, IFIP CONGRESS 1974, 386-393, North-Holland, Amsterdam.

[6]. K. Howell: “First Computer Language”, JCSC 18, 4, April 2003

[7]. First Course Language for Computer Science majors ftp://ftp.cps.msu.edu/pub/arch/CS1_Language_List/ Retrieved April, 2012

[8]. C. Jones: Applied Software Measurement: Assuring Productivity and Quality, Second Edition, McGraw-Hill 1996

[9]. E.W. Dijkstra: The Humble Programmer, CACM, 15, 10, 1972.

[10]. A. Sutton and Bjarne Stroustrup: Design of Concept Libraries for C++. Proc. SLE 2011 (International Conference on Software Language Engineering). July 2011

[11]. T. Wang, X. Su, P. Ma, Y. Wang, and K. Wang. 2011. Ability-training-oriented automated assessment in introductory programming course. Comput. Educ. 56, 1: 220-226, 2011.

[12]. B. Stroustrup. 2009. Programming in an undergraduate CS curriculum. In *Proceedings of the 14th Western Canadian Conference on Computing Education* (WCCCE '09), Roelof Brouwer, Diana Cukierman, and George Tsiknis (Eds.). ACM, New York, NY, USA, 82-89.

[13]. W.M. Kunkle (2010) The Impact of Different Teaching Approaches and Languages on Student Learning of Introductory Programming Concepts.(Doctoral dissertation). http://hdl.handle.net/1860/3380.

[14]. ACM/IEEE-CS Joint Interim Review Task Force. 2008. Computer Science Curriculum 2008: An Interim Revision of CS 2001, Report from the Interim Review Task Force.
http://www.acm.org/education/curricula/ComputerScience2008.pdf