MEDWELL PUBLISHING
Research Journal of Applied Sciences 4 (1): 41-49, 2009
ISSN: 1815-932X
© Medwell Journals, 2009

Introducing Object Oriented Programming in Tertiary Institutions: Which Language Is Most Appropriate?

Godspower O. Ekuobase, Veronica V.N. Akwukwuma and Francesca A. Egbokhare
Department of Computer Science, University of Benin, Benin City, Nigeria

Abstract: Object Oriented Programming (OOP) has attained a level of acceptance in the software development community so much so that one is now considered a charlatan to have a degree in computing without the flare for OOP. Computing students must therefore, be equipped with this skill. The need to do this using the most appropriate language is critical. We are not aware that any such language has been identified based on scientific instruments or parameters, except on individual opinion or affiliation. To this end, adopting the enhanced Alexander's scheme, we identified and defined a set of criteria critical to the selection of the most appropriate programming language for introducing students to Object-Oriented concepts and programming. Furthermore, based on these criteria, we produced a rating showing the appropriateness of selected programming languages to the teaching and learning of OOP.

Key words: OOP, teaching and learning, programming languages, selection criteria, indiviual opinion, affiliation

INTRODUCTION

It is true that computing is not programming but programming, as we know, is to computing what leaven is to bread. Is it any wonder that one cannot have a formal degree in computing without taking and passing courses in programming? The art of programming is often taught using a particular programming language though with the misconception on the part of some instructors that the skill called programming is the same as mastering a programming language or two. Several programming languages abound and can be categorized in several ways. One of such ways is the programming paradigm they enforce or support. We have several programming paradigms that are still active which include: functional, modular, structured, object oriented, agile and open source (Ekuobase, 2006; Raccoon, 1997). Object-oriented paradigm, though relatively new, has attained a high level of acceptance in the software development community so much so that one is now considered a charlatan to have a degree in computing without some good abilities in Object Oriented programming. It is therefore, imperative we equip computing students with this skill.

Clearly, this can best be done using languages that enforce or at least support OOP. These OOP languages in existence are plentiful (Sebesta, 2008) and it is therefore, not possible to use all of them. The question now is which of the programming languages should be used to introduce students to OOP? The answer is simple, the most appropriate programming language. Using just any (OOP) programming language will push us into the dilemma of the Flon (1975) axiom, which for example, in the extreme any way, supports the use of assembly language in distributed computing. We are not aware of any work that has scientifically justified any programming language as the best OOP language for teaching and learning OOP paradigm. We are however, aware of programming language evaluation criteria but often frustrated by the lack of metric or scheme that will enable us single out a particular language as the most appropriate in a given context or domain, particularly, with several alternatives.

This study identifies and defines a set of criteria for choosing the most appropriate language for the teaching and learning of OOP and adopted a rating scheme for it. The adopted scheme is the enhanced Alexander scheme (Onibere and Ekuobase, 2006; Ekuobase and Onibere, 2007) with the understanding that the problem of selecting the most appropriate software process model for a given software development project is similar in nature to our research question. In particular, we produced a rating for some selected programming languages using these criteria with the adopted scheme. The rating shows the suitability of these languages to the teaching and learning of OOP. The programming languages selected were the programming languages rated among the IT skills required in a tough job market: Java, C#, C++, C and Visual BASIC (Prabhakar *et al*., 2005); since students are basically trained for the job market.

Corresponding Author: Godspower O. Ekuobase, Department of Computer Science, University of Benin, Benin City, Nigeria

This research assumes familiarity with these programming languages as well as with the Object Oriented paradigm.

OOP LANGUAGE SELECTION CRITERIA

Based on available literatures, Parker *et al.* (2006), Armstrong (2006), Obasanjo (2005), Berge *et al.* (2005), De Raadt *et al.* (2003), Capretz (2003), McIver (2002), McIver and Conway 1996), Ghezzi and Jazayeri (1998), Howland (1997), Kolling *et al.* (1995), Meyer (1993), Schneider (1978), Stroustrup (1987) and Tharp (1982), experience and consultations with selected programming language experts, lecturers and students well grounded in at least one of our language of interest, we identified thirty five criteria that are critical to the choice of a language for teaching and learning OOP. This final criteria set is the result of scrutinized preliminary criteria set by a team of programming language experts and senior faculty members in the Department of Computer Science, University of Benin and in one of the Information Technology and Mathematical Modelling sessions of the International Conference on Advances in Engineering and Technology held in 2006 at Entebbe-Uganda. These criteria, for ease of understanding and adaptation, are partitioned into 4 categories, namely social-economic attributes, methodology attributes, language attributes and universality attributes. For example, to adapt this scheme to, say, structured programming, all we need do is to identify attributes peculiar to structured programming methodology and use them as the methodology attributes instead; since the ones in this study are unique to OOP. These attributes are discussed in the following subsections and summarized in Table 1. We only describe criterion values which otherwise are trivial, when their meanings are ambiguous or not explicit.

Socio-economic attributes: These are the non-technical characteristics of a programming language that have a direct impact on the teaching and learning of that language with no Software Development Methodology (SDM) taking into consideration. These attributes and their values are discussed below. They include: language size, Familiarity, expressivity, compiler cost, compiler availability, text availability, cost of textbooks, Instructor's expertise, ease of compilation and industrial demand.

Language Size (C1): This is the size of the entire programming language specification. Values are: V1 = (negligible, small, moderate, large, very-large, extra-large).

Familiarity (C2): By this we mean that the language uses notations that are similar to existing and common language notations. Values are: V2 = (none, slight, moderate, significant, substantial, extreme).

Expressivity (C3): How easily does the language allow the programmer naturally express itself in the domain of problem formulation? Values are: V3 = (intractable, complex, restricted, simple, flexible, seamless).

Compiler cost (C4): This measures the mode of the cost of the available standard compilers for the language. Different vendors normally have different prices for their products. Values are: V4 = (free, negligible, cheap, affordable, expensive, outrageous). All proprietary software is at best cheap.

Compiler availability (C5): Here, we ask the question, are the compilers for the language easy to come by? Values are: V5 = (not available, scarce, limited, adequate, ample, abundant).

Instructor's expertise (C6): This measures the knowledge and experience of available instructor's in a given programming language. Values are: V6 = (novice, comfortable, knowledgeable, experienced, expert, experienced-expert).

Cost of textbooks (C7): This measures the mode of the cost of the standard textbooks or resource available for the language. Values are: V7 = (free, negligible, cheap affordable, expensive, outrageous).

Text availability (C8): This is the degree to which standard accounts of a particular programming language is obtainable. Values are: V8 = (not available, scarce, limited, adequate, ample, abundant).

Industrial demand (C9): This is the level of need or desire for a particular programming language in the available market (industry). Values are: V9 = (negligible, low, moderate, substantial, very-high, extreme).

Ease of compilation (C10): Here, we measure the convenience of language compilation i.e. how easy it is to get the source code to the language compiler for processing into a target code. Values are: V10 = (trivial, simple, demanding, difficult, complex, intractable).

Methodology attributes: These are features very dear to the Object Oriented SDM (OO-SDM). They help distinguish the OO-SDM from other SDM. Consequently,

Table 1: Language selection criteria with 6 point values each

	Function point values					
Criteria Cij	Vi1	Vi2	Vi3	Vi4	Vi5	Vi6
Language size	Negligible	Small	Moderate	Large	Very-large	Extra-large
Familiarity	None	Slight	Moderate	Significant	Substantial	Extreme
Expressivity	Intractable	Complex	Restricted	Simple	Flexible	Seamless
Compiler cost	Free	Negligible	Cheap	Affordable	Expensive	Outrageous
Compiler availability	Not available	Scarce	Limited	Adequate	Ample	Abundant
Instructor's expertise	Novice	Comfortable	Knowledgeable	Experienced	Expert	Experienced-expert
Cost of text books	Free	Negligible	Cheap	Affordable	Expensive	Outrageous
Text availability	Not available	Scarce	Limited	Adequate	Ample	Abundant
Industrial demand	Negligible	Low	Moderate	Substantial	Very high	Extreme
Ease of compilation	Trivial	Simple	Demanding	Difficult	Complex	Intractable
Inheritance support	None	Slight	Moderate	Significant	Substantial	Extreme
Support for reuse	None	Slight	Moderate	Significant	Substantial	Extreme
Polymorphism support	None	Slight	Moderate	Significant	Substantial	Extreme
Modifier support	None	Negligible	Significant	Substantial	Extreme	Imposed
Class support	None	Negligible	Significant	Substantial	Extreme	Imposed
OOAD support	None	Slight	Moderate	Significant	Substantial	Extreme
GUI capability	None	Negligible	Significant	Substantial	Extreme	Imposed
Encapsulation support	None	Slight	Moderate	Significant	Substantial	Extreme
Exception handling capability	None	Slight	Moderate	Significant	Substantial	Extreme
Syntax complexity	Trivial	Simple	Demanding	Difficult	Complex	Intractable
Orthogonality	Indifferent	Negligible	Significant	Substantial	Extreme	Imposed
I/O capability	Trivial	Simple	Demanding	Difficult	Complex	Intractable
Typing	Not typed	Static	Dynamic	Flexible	Weak	Strong
Compiler capability	Interprets slowly	Interprets efficiently	Flexibly slow	Flexibly fast	Compiles slowly	Compiles efficiently
Support for user defined data types	None	Negligible	Significant	Substantial	Extreme	Enforced
Support for data structures	None	Slight	Moderate	Significant	Substantial	Extreme
Pointer capability	None	Negligible	Significant	Substantial	Extreme	Enforced
Usability of library functions	Harsh	Slight	Moderate	Friendly	Very friendly	Seamless
Compilation support tools	None	Negligible	Poor	Moderate	Substantial	Excellent
Memory management features	None	Negligible	Poor	Moderate	Substantial	Excellent
Operating system support	None	Negligible	Poor	Moderate	Substantial	Excellent
Library capability	None	Negligible	Poor	Moderate	Substantial	Excellent
Networking support	None	Negligible	Poor	Moderate	Substantial	Excellent
Portability	None	Negligible	Poor	Moderate	Substantial	Excellent
Domain support	None	Negligible	Few	Many	Most	Any

how well a language enforces these attributes defines its OO capacity. These attributes and their values are discussed. They include inheritance support, support for reuse, polymorphism support, modifier support, support for class, Object Oriented Analysis and Design (OOAD) support, Graphical User Interface (GUI) capability, encapsulation support and exception handling capability.

Inheritance support (C11): Our interest here is how well the language supports the concept of inheritance. Values are: V11 = (none, slight, moderate, significant, substantial, extreme). We say it is extreme if the language specification makes it intractable for programmers to do without the attribute when using the language.

Support for reuse (C12): Ease of adapting existing program or program segments to new goals. Values are: V12 = (none, slight, moderate, significant, substantial, extreme).

Polymorphism support (C13): How well the language supports the concept of polymorphism. Values are: V13 = (none, slight, moderate, significant, substantial, extreme).

Modifier support (C14): A common element of the OOP languages is the presence of access modifiers, indicating different levels of class encapsulation. Values are: V14 = (none, negligible, significant, substantial, extreme, imposed).

Class support (C15): How well the language supports the class concept. Values are: V15 = (against, negligible, significant, substantial, extreme, imposed).

OOAD support (C16): How well the language supports the view that everything is an object. Values are: V16 = (none, slight, moderate, significant, substantial, extreme).

GUI capability (C17): The features in the language which allow programmers to easily and flexibly build programs with graphical interfaces that aid quite

significantly the usability of the program. Values are: V17 = (none, negligible, significant, substantial, extreme, imposed).

Encapsulation support (C18): This is the ease and flexibility of using/building objects property and functionality in a language without any prior knowledge about the objects detail. Values are: V18 = (none, slight, moderate, significant, substantial, extreme).

Exception handling capability (C19): The features or constructs in the language that allow a programmer to easily and flexibly handle exception thereby making programs more robust. Values are: V19 = (none, slight, moderate, significant, substantial, extreme).

Language attributes: These are the technical characteristics of a programming language that have a direct impact on the learning and teaching of the language without any consideration for a particular SDM. These attributes and their values are discussed. They include syntax complexity, orthogonality, I/O complexity, construct reliability, typing, compiler capability, support for user defined data types, support for data structures, pointer capability, restricted aliasing, usability of library function, compilation tool support and memory management feature.

Syntax complexity (C20): This criterion assesses the intricate details involved in the rules governing which statements and combinations of statements in a programming language will be acceptable to a compiler for that language. Values are: V20 = (trivial, simple, demanding, difficult, complex, intractable).

Orthogonality (C21): This measures how well the language resists actions that can generate side effects in programs. Values are: V21 = (indifferent, negligible, significant, substantial, extreme, imposed). We say it is imposed when programmers cannot write code with side effects in the language i.e. this attribute is forced on the programmer in the language.

I/O complexity (C22): This criterion describes the ease and flexibility with which a programmer can accept input data into a program. Values are: V22 = (trivial, simple, demanding, difficult, complex, intractable).

Typing (C23): Here, we measure both the nature and degree of typing of a programming language. Values are: V23 = (not typed, static, dynamic, flexible, weak, strong). We say it is flexible when it binds both statically and dynamically.

Compiler capability (C24): This measures the nature and efficiency of a compiler. Values are: V24 = (interprets slowly, interprets efficiently, flexibly slow, flexibly fast, compiles slowly, compiles efficiently). Flexibility here means having both interpretation and compilation capability. For example, interprets efficiently will be ideal for teaching and learning any language.

Support for user defined data types (C25): This criterion measures the degree to which a programmer can define his own data types to aid better expressivity in programming.

Values are: V25 = (none, negligible, significant, substantial, extreme, enforced).

Support for data structures (C26): This criterion deals with the ability of a language to support a variety of data values such as integers, strings, real, set, list etc. Values are: V26 = (none, slight, moderate, significant, substantial, extreme).

Pointer capability (C27): This criterion measures the ease and flexibility with which programmers can use indirect methods in accessing memory. Values are: V27 = (none, negligible, significant, substantial, extreme, enforced).

Usability of library functions (C28): This measures the degree to which the library functions can be conveniently put to use. Values are: V28 = (harsh, slight, moderate, friendly, very-friendly, seamless).

Compilation tool support (C29): Here, we are interested in the compiler support tools such as debuggers and lexical analyzers in most of the available language compilers. Values are: V29 = (none, negligible, poor, moderate, substantial, excellent).

Memory management features (C30): This measures the language features or capabilities that allow programmers make efficient use of computer memory.

Values are V30 = (none, negligible, poor, moderate, substantial, excellent). Excellent here means that memory management is done automatically by the compiler.

Universality attributes; These are tools or technology that support or are supported by a given programming language that encourage a wide spread acceptability, applicability or use of the language. These attributes and their values are discussed below. They include operating system support, library capability, networking support, compilation tools support, portability and domain support.

Operating systems support (C31): This criterion describes the degree to which a compiled program in programming language can switch between several operating system platforms. Values are: V31 = (none, negligible, poor, moderate, substantial, excellent).

Library capability (C32): We want to ask ourselves, how equipped is the programming language library. Values are: V32 = (none, negligible, poor, moderate, substantial, excellent).

Networking support (C33): How well does the language support network (LAN and WAN) programming. Values are: V33 = (none, negligible, poor, moderate, substantial, excellent).

Portability (C34): This criterion describes the degree to which an executable program in programming language can switch between several hardware platforms. Values are: V34 = (none, negligible, poor, moderate, substantial, excellent).

Domain support (C35): Here, our interest is the number of problem domain the programming language can be effectively and conveniently used. It measures the degree of versatility of a programming language. Values are: V35 = (none, negligible, few, many, most, any).

The Alexander's scaling scheme: The Alexander's scaling scheme as presented in Onibere and Ekuobase (2006) is as follows:

Step 1: Examine the project's attributes and determine Sij's for each criterion that best describe the project.

Step 2: For each process model, compute:

$$\text{Rating} = \sum_{i=1}^{20}\sum_{i=1}^{3}(S_{ij}\times a_{ij}) \quad (1)$$

Thus, for the modified scheme (Onibere and Ekuobase, 2006) we have:

$$\text{Rating} = \sum_{i=1}^{27}\sum_{i=1}^{6}(S_{ij}\times a_{ij}) \quad (2)$$

Step 3: The process model with the highest RATING is the best choice, 2nd highest is the 2nd best choice, etc.

The notations used were as used in Onibere and Ekuobase (2006):

- C = The set of 20 criteria with elements C_i, i = 1.20 and 1.27 for the Alexander and modified set of criteria, respectively
- V_i = The ordered vector of elements V_{ij} of values that each criterion C_i may take on e.g. for the modified set of criteria, V11-V16 have values novice, knowledgeable, experienced, well-experienced, expert and experienced expert, respectively
- S_{ij} = A binary indicator of which value of criterion Ci applies to the particular project. The S_{ij}'s form a project characteristics matrix that reflects the attributes of a project
- a_{ij} = The applicability of a process model to a V_{ij} e.g. a 26 = 1 for Waterfall model indicates that the process is appropriate for projects having users with descriptive ability to express needs. The aij's for each process model form a matrix used to determine the rating of a process model with respect to a particular project

To adapt the enhanced scheme to our case, we have,

Step 1: Examine the attributes for the teaching and learning of OOP and determine Sij's for each criterion that best describe it.

Step 2: For each programming language, compute:

$$\text{Rating} = \sum_{i=1}^{35}\sum_{i=1}^{6}(S_{ij}\times a_{ij}) \quad (3)$$

Step 3: The programming language with the highest RATING is the best choice, 2nd highest is the 2nd best choice, etc. Where the following notations hold as follows:

- C = The set of 35 criteria with elements Ci, i = 1..35

V_i = The ordered vector of elements V_{ij} of values that each criterion Ci may take on e.g., for this set of criteria, V11-V16 have values negligible, small, mo-derate, large, very-large, extra-large, respectively

S_{ij} = A binary indicator of which value of criterion Ci applies to the teaching and learning of OOP. The Sij's form a project characteristics matrix that reflects the attributes of this task

a_{ij} = The suitability of a programming language to a Vij e.g., a65 = 1 for C indicates that the language is suitable for the teaching and learning of OOP if the college have an instructor that is an expert in C programming. The aij's for each programming language form a matrix used to determine the rating of a programming language with respect to the teaching and learning of OOP

Selection matrices: This study contains the standard template that best describes the criteria values for teaching and learning OOP in an introductory class (Table 2), as well as those of the 5 selected languages (Table 3-5). Known experts in the individual languages and some selected students of the Department of Computer Science, University of Benin and Computer and Information Technology Department of Bowen University both in Nigeria were consulted separately and the Table 3-5 are a reflection of popular evaluation (i.e., we took the choice mode).

In doing this, we were driven by the ease of teaching and learning OOP concepts and programming, within the time frame of a semester (4 h of teaching) to computing students of University of Benin and Bowen University with no prior programming skills or experience.

The next step, called the matching process, defines the appropriateness of a given process model to a given project. This is done by computation using the Eq. (3). If this is done correctly, simple percentage (Rating * 100/max. Rating) for each selected programming language will give a nod to the values in Table 6.

Table 2: Selection matrix for teaching and learning OOP in University of Benin (UNIBEN), Nigeria

Criteria	Vi1	Vi2	Vi3	Vi4	Vi5	Vi6
Language size	0	1	1	0	0	0
Familiarity	0	0	0	1	1	1
Expressivity	0	0	1	1	0	0
Compiler cost	1	1	1	0	0	0
Compiler availability	0	0	0	1	1	1
Instructor's expertise	0	0	0	0	1	1
Cost of text books	1	1	1	0	0	0
Text availability	0	0	0	1	1	1
Industrial demand	0	0	0	1	1	1
Ease of compilation	1	1	0	0	0	0
Inheritance support	0	0	0	0	0	1
Support for reuse	0	0	0	0	0	1
Polymorphism support	0	0	1	1	0	0
Modifier support	0	0	1	1	1	0
Class support	0	0	0	0	0	1
OOAD support	0	0	0	0	0	1
GUI capability	0	1	1	1	1	0
Encapsulation support	0	0	0	1	1	1
Exception handling capability	0	0	0	1	1	0
Syntax complexity	1	1	0	0	0	0
Orthogonality	0	0	0	0	1	1
I/O capability	1	1	0	0	0	0
Typing	0	1	0	0	0	1
Compiler capability	1	1	1	1	0	0
Support for user defined data types	0	0	0	1	1	0
Support for data structures	0	0	0	0	1	1
Pointer capability	0	0	1	1	0	0
Usability of library functions	0	0	0	0	1	1
Compilation support tools	0	0	0	1	1	1
Memory management features	0	0	0	0	1	1
Operating system support	0	0	0	0	1	1
Library capability	0	0	0	0	1	1
Networking support	0	0	0	0	1	1
Portability	0	0	0	0	1	1
Domain support	0	0	0	1	1	1

Table 3: Selection matrices for teaching and learning OOP using C# and visual basic in UNIBEN, Nigeria

Languages	C#						Visual basic					
criteria	V1	V2	V3	V4	V5	V6	V1	V2	V3	V4	V5	V6
Language size	0	1	0	0	0	0	0	0	1	0	0	0
Familiarity	0	0	0	1	0	0	0	0	0	0	1	0
Expressivity	0	0	0	1	0	0	0	0	0	0	1	0
Compiler cost	0	0	1	0	0	0	0	0	0	1	0	0
Compiler availability	0	1	0	0	0	0	0	0	0	1	0	0
Instructor's expertise	1	0	0	0	0	0	0	0	1	0	0	0
Cost of text books	0	0	0	0	1	0	0	0	0	1	0	0
Text availability	0	1	0	0	0	0	0	0	0	1	0	0
Industrial demand	0	1	0	0	0	0	0	0	0	0	1	0
Ease of compilation	0	1	0	0	0	0	1	0	0	0	0	0
Inheritance support	0	0	0	0	0	1	1	0	0	0	0	0
Support for reuse	0	0	0	0	0	1	0	0	0	1	0	0
Polymorphism support	0	0	0	1	0	0	0	1	0	0	0	0
Modifier support	0	0	0	0	1	0	0	0	1	0	0	0
Class support	0	0	0	0	0	1	0	0	0	1	0	0
OOAD support	0	0	0	0	0	1	0	1	0	0	0	0
GUI capability	0	0	1	0	0	0	0	0	0	0	0	1
Encapsulation support	0	0	0	0	1	0	0	0	0	1	0	0
Exception handling capability	0	0	0	0	1	0	0	0	0	0	1	0
Syntax complexity	0	0	0	1	0	0	0	1	0	0	0	0
Orthogonality	0	0	0	0	1	0	0	0	0	0	1	0
I/O capability	0	1	0	0	0	0	0	1	0	0	0	0
Typing	0	1	0	0	0	0	0	0	0	1	0	0
Compiler capability	0	0	0	1	0	0	0	0	0	0	0	1
Support for user defined data types	0	0	0	0	1	0	0	0	0	1	0	0
Support for data structures	0	0	0	0	1	0	0	0	0	1	0	0
Pointer capability	0	0	1	0	0	0	0	1	0	0	0	0
Usability of library functions	0	0	0	0	1	0	0	0	0	0	1	0
Compilation support tools	0	0	0	0	1	0	0	0	0	0	0	1
Memory management features	0	0	0	0	0	1	0	0	0	0	0	1
Operating system support	0	0	1	0	0	0	0	1	0	0	0	0
Library capability	0	0	0	0	0	1	0	0	0	0	1	0
Networking support	0	0	0	0	0	1	0	0	0	0	1	0
Portability	0	0	0	1	0	0	0	0	0	1	0	0
Domain support	0	0	0	0	1	0	0	0	0	1	0	0

Table 4: Selection matrices for teaching and learning OOP using C++ and Java in UNIBEN, Nigeria

Languages	C++						Java					
Criteria	V1	V2	V3	V4	V5	V6	V1	V2	V3	V4	V5	V6
Language size	0	0	0	0	1	0	0	1	0	0	0	0
Familiarity	0	0	0	0	1	0	0	0	0	1	0	0
Expressivity	0	0	1	0	0	0	0	0	1	0	0	0
Compiler cost	0	0	0	1	0	0	0	1	0	0	0	0
Compiler availability	0	0	0	1	0	0	0	0	1	0	0	0
Instructor's expertise	0	0	1	0	0	0	0	1	0	0	0	0
Cost of text books	0	0	0	1	0	0	0	0	0	0	1	0
Text availability	0	0	0	1	0	0	0	0	1	0	0	0
Industrial demand	0	0	0	1	0	0	0	0	0	1	0	0
Ease of compilation	0	1	0	0	0	0	0	0	1	0	0	0
Inheritance support	0	0	0	0	0	1	0	0	0	0	0	1
Support for reuse	0	0	0	1	0	0	0	0	0	0	0	1
Polymorphism support	0	0	0	0	1	0	0	0	1	0	0	0
Modifier support	0	0	0	0	1	0	0	0	0	0	1	0
Class support	0	0	0	1	0	0	0	0	0	0	0	1
OOAD support	0	0	0	1	0	0	0	0	0	0	0	1
GUI capability	1	0	0	0	0	0	0	0	0	1	0	0
Encapsulation support	0	0	0	1	0	0	0	0	0	0	1	0
Exception handling capability	0	0	0	1	0	0	0	0	0	0	0	1
Syntax complexity	0	0	1	0	0	0	0	0	0	0	1	0
Orthogonality	0	1	0	0	0	0	0	0	0	0	1	0
I/O capability	0	1	0	0	0	0	0	0	1	0	0	0
Typing	0	1	0	0	0	1	0	1	0	0	0	1
Compiler capability	0	0	0	0	0	1	1	0	1	0	0	0
Support for user defined data types	0	0	0	0	1	0	0	0	0	1	0	0

Table 4: Continue

Languages	C++						Java					
criteria	V1	V2	V3	V4	V5	V6	V1	V2	V3	V4	V5	V6
Support for data structures	0	0	0	0	0	1	0	0	0	0	1	0
Pointer capability	0	0	0	0	1	0	0	1	0	0	0	0
Usability of library functions	0	0	1	0	0	0	0	0	0	1	0	0
Compilation support tools	0	0	0	0	1	0	0	0	0	0	1	0
Memory management features	0	0	1	0	0	0	0	0	0	0	0	1
Operating system support	0	0	0	0	1	0	0	0	0	0	1	0
Library capability	0	0	0	0	0	1	0	0	0	0	0	1
Networking support	0	0	0	0	1	0	0	0	0	0	1	0
Portability	0	0	0	1	0	0	0	0	0	0	1	0
Domain support	0	0	0	1	0	0	0	0	0	0	1	0

Table 5: Selection matrices for teaching and learning OOP using C in UNIBEN, Nigeria

Languages	C					
Criteria	V1	V2	V3	V4	V5	V6
Language size	0	0	0	1	0	0
Familiarity	0	0	0	1	0	0
Expressivity	0	0	0	1	0	0
Compiler cost	0	1	0	0	0	0
Compiler availability	0	0	1	0	0	0
Instructor's expertise	0	0	1	0	0	0
Cost of text books	0	0	0	1	0	0
Text availability	0	0	1	0	0	0
Industrial demand	1	0	0	0	0	0
Ease of compilation	0	1	0	0	0	0
Inheritance support	1	0	0	0	0	0
Support for reuse	0	1	0	0	0	0
Polymorphism support	1	0	0	0	0	0
Modifier support	0	0	1	0	0	0
Class support	1	0	0	0	0	0
OOAD support	1	0	0	0	0	0
GUI capability	1	0	0	0	0	0
Encapsulation support	0	0	1	0	0	0
Exception handling capability	0	0	1	0	0	0
Syntax complexity	0	1	0	0	0	0
Orthogonality	0	0	1	0	0	0
I/O capability	0	0	1	0	0	0
Typing	0	1	0	0	0	0
Compiler capability	0	0	0	0	0	1
Support for user defined data types	0	0	0	0	1	0
Support for data structures	0	0	0	0	1	0
Pointer capability	0	0	0	0	1	0
Usability of library functions	0	0	0	1	0	0
Compilation support tools	0	0	0	0	1	0
Memory management features	0	0	1	0	0	0
Operating system support	0	0	0	0	1	0
Library capability	0	0	0	0	0	1
Networking support	0	0	0	0	1	0
Portability	0	0	0	0	1	0
Domain support	0	0	0	1	0	0

Table 6: Suitability of languages in teaching and learning OOP in an introductory level

Language	Rank (%)	Position
C#	77.1	1st
Java	74.3	2nd
C++	57.2	3rd
Visual basic	54.3	4th
C	45.7	5th

RESULTS AND DISCUSSION

From Table 6, it is obvious that none OOP languages are inappropriate for teaching and learning OOP and that a pure OOP language is an excellent choice in the teaching and learning of OOP. Furthermore, our experiment shows that C# is the most appropriate Object Oriented Programming language for teaching and learning OOP in an introductory programming class.

During the course of this research, many of our consultants wondered why C was selected at all to be considered for the experiment. We always replied that it will help show the weakness of our selection procedure. For example, if the result from the experiment rated the C language above, say, C++ then it would have been a funny result. Clearly, our result shows that the selection scheme is strong and reliable.

Very importantly, C# is expected to do better in the more developed countries of the world since it fell to Java particularly in Criterion 8 and 9 in Nigeria. This may not likely be the case in the United States or in Europe and this is capable of increasing the suitability rating of C# by up to 5.7% in these environments.

CONCLUSION

A scheme now exists for choosing the most appropriate programming language for teaching and learning OOP in tertiary institutions. It is the duty of Faculty in consultation with students and experts in programming languages, OOP in particular, to determine the various matrices in making a choice for their institutions. This may not necessarily be done annually. Our result shows that core OOP languages are the most appropriate language for teaching and learning OOP; with C# standing out for both the new and the old universities in Nigeria.

RECOMMENDATIONS

Consequently, we wish to recommend that efforts be put into making C# more attractive to the academia than it is presently enjoying in Nigeria in the form of availability of test, staff training and curriculum development.

ACKNOWLEDGEMENT

We are indeed grateful to the academic and programming staff of Computer Science Department, University of Benin and CIT Department, University of Bowen for their assistance in building the selection scheme's input data (i.e. Appendix). To members of the Information Technology and Mathematical Modelling session of the conference on Advances in Engineering and Technology held in Entebbe-Uganda, July 2006 who helped in the scrutiny of the selection criteria, we cannot thank you enough.

REFERENCES

Armstrong, D., 2006. The quarks of object oriented development. Commun. ACM, 49 (2): 123-128.

Berge, O., R.E. Borge and A. Fjuk, 2005. Learning Object Oriented Programming. http: //www.ifi.uio.no.

Capretz, L.F., 2003. A brief history of the object-oriented approach. ACM Soft. Eng. Notes, 28 (2): 1-10.

De Raadt, M., R. Watson and M. Toleman, 2003. Introductory Programming Language at Australian Universities at the beginning of the 21st century. J. Res. Pract. Inform. Technol., 35 (3): 163-167.

Ekuobase, G.O., 2006. Software Creative Milestones, International Conference on Advances in Engineering and Technology (AET). Entebbe-Uganda, Elsevier, pp: 848-855.

Ekuobase, G.O. and E.A. Onibere, 2007. Software Process Selection Criteria in Perspective. Int. J. Phys. Sci., 2 (3): 81-89.

Flon, L., 1975. On research in structured programming,. ACM SIGPLAN Notices,10 (10): 16-17.

Ghezzi, C. and M. Jazayeri, 1997. Programming Language Concepts. 3rd Edn. NY: John Wiley and Sons, pp: 448.

Howland, J.E., 1997. It's all in the Language: Yet another look at the choice of programming language for teaching computer science. J. Comput. Small Colleges, 12 (4): 58-74.

Kolling, M., B. Koch and J. Roseenberg, 1995. Requirements for a 1st year object oriented teaching language. ACM SIGCSE Bull., 27 (1): 173-177.

McIver, L., 2002. Evaluating languages and environments for novice programmers. Proc. of 14th Annual Meeting of the Psychology of Programming Interest Group, London, pp: 100-110.

McIver, L. and D.M. Conway, 1996. Seven deadly sins of introductory programming language design. Proc. Software Engineering: Education and Practice, CA: USA. IEEE Computer Society Press, pp: 309-316.

Meyer, B., 1993. Towards an object oriented curriculum. J. Object Oriented Program., 6 (2): 76-81.

Obasanjo, D., 2005. A comparison of Microsoft's C# Programming Language to Sun Microsystems Java Programming Language. http://www.25hoursaday. com/CsharpVsJava.html.

Onibere, E.A. and G.O. Ekuobase, 2006. Enhanced Software Process Selection Criteria. J. Inst. Math. Comput. Sci., 17 (1): 17-32.

Parker, K.R., T.A. Ottaway and J.T. Chao, 2006. Criteria for the Selection of a programming language for introductory courses. Int. J. Knowledge Learning, 2 (1 and 2): 119-139.

Prabhakar, B., C.R. Litecky and K. Arnette, 2005. IT Skills in a tough job market. Commun. ACM, 48 (10): 91-94.

Raccoon, L.B.S., 1997. Fifty years of Progress in Software Engineering, Software Engineering Notes. ACM SIGSOFT, 22 (1): 88-104.

Schneider, G.M., 1978: The introductory programming course in computer science: Ten principles, ACM SIGCSE Bull., 10 (1): 107-114.

Sebesta, R.W., 2008. Concepts of Programming Languages. 8th Edn. Addison-Wesley.

Stroustrup, B., 1987. What is Object Oriented Programming? Springer-Verlag. Lecture Notes Comput. Sci. Series, 276: 51-70.

Tharp, A.L., 1982. Selecting the 'right' programming language. ACM SIGCSE Bull., 14 (1): 151-155.