IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661,p-ISSN: 2278-8727, Volume 26, Issue 5, Ser. 1 (Sept. – Oct. 2024), PP 26-32
www.iosrjournals.org

Demystifying The Object-Oriented Features Of Popular Object-Oriented Programming Languages.

Fergus U Onu[1], Ifeoma Catherine Oliver[2], Mamah Chukwurah Hyginus[3], Afolabi Idris Yinka[4], Nwodo John O[5], Yunisa Sunday[6]
(Computer Science Department, Ebonyi State University, Abakaliki, Ebonyi State, Nigeria).
(Akanu Ibiam Federal Polytecnic Unwana, Ebonyi State, Nigeria)
(Alex Ekwueme Federal University Ndufu Alike, Ikwo, Ebonyi State, Nigeria)
(Scientific Equipment Development Institute (Sedi-E) Enugu, Enugu State,Nigeria).
(Confluence University Of Science And Technology, Osara, Kogi State, Nigeria)

Abstract
Background: *Indeed, Object-Oriented Programming (OOP) has emerged as one of the major fundamental paradigms in today's world of software development as it contains principles that enhance modularity, reusability, and maintainability of software. This article explores deeper into some of the facets of OOP aspects as exhibited in some of the most frequently used programming languages ranging from Java, Python, C++, and C# with an intent of offering substantive information to aspects like encapsulation, inheritance, polymorphism, and abstraction. The article synthesizes the ones focusing on the relevance and conceptualization of OOP and problems related to it in the context of learners and practitioners. It also describes strengths and weaknesses of using C++, Python, and Java in relation to OOP while stressing peculiarity of these languages. Some of the core principles of OOP are highlighted such as encapsulation, inheritance, polymorphism, and abstraction and how they are complicated and significant in today's software engineering.*
Materials and Methods: *There was a presentation of a survey on which the research is based to determine the problems that programmers encounter when applying OOP features using a sample of 80 programmers in the computer science fields in some Nigerian Universities.*
Results*: Findings reveal the significance of understanding OOP concepts in popular languages and suggest ways of increasing developers' efficiency, minimizing errors, and improving the quality of developed software.*
Conclusion*: Suggestions include a necessity for training and education to enhance software developers' understanding of OOP features. The findings can guide the creation of training materials and courses centered on OOP concepts, aiming to improve software development practices and elevate the quality of software products.*
Keyword: *Object-Oriented Programming, Encapsulation, Inheritance, Polymorphism, Abstraction.*

Date of Submission: 03-09-2024 Date of Acceptance: 13-09-2024

I. Introduction

In the constantly developing field of the software creation, one of the most fundamental and basis paradigms is the OOP, based on the principles of modularity, reusability, and maintainability. Java, Python, C++, and C# are among the widely used languages that include OOP as a key element to offer programmers effective means for creating stable and efficient software solutions. These object oriented programming languages constitute the back end programming languages used in the creation of majority of artificial intelligent that is now revolutionizing the world [1]. [17] Has claimed that object-oriented techniques have found more acceptance in software development and engineering. Nevertheless, these languages' object-oriented features are complex, which makes it difficult for developers to delve deeper into their workings when in search of information. Object-oriented processing is the utilization of developed program quilt units, objects, classes, and subclasses, etc., to avoid complicated formula manufacture and easily maintain [6]. The purpose of this article is to explain these languages' object-oriented characteristics and to explain and demystify many of the complexities surrounding them while giving the reader an insight into the processes behind OOP. Hence, the features such as encapsulation, inheritance, polymorphism, and abstraction are explained to enable a developer to achieve the optimum utilization of OOP in their projects. There is a need to stress the importance of understanding the subtleties of the features provided by object-oriented programming paradigm. However, this understanding also improves quality and scalability of the code as well as provides developers with better skills in creating efficient and easy to maintain software systems.

The aim of this study is to specifically analyze the aspects of Object-oriented features of Java, Python, C++ and C# looking at how each of the languages is implementing OOP concepts in its own way. Our intention delivering the idea practically with examples and comparing with the best and other options will help the developers to make effective decisions while choosing and applying the subsets of object-oriented design patterns. Other researchers that have also recognized the significance of object oriented software reusability include more recent researchers among them being [5] and [4]. Our goal is to present the various aspects of object-oriented features in today's programming languages to benefit both the introductory and advanced programmers by offering them improved skills to transform their coding abilities.

II. Literature Review

Object Oriented Programming is one of the applied languages that operates from the basic principle of objects in fields, codes or procedures. Multi-paradigm OOP language is backed up by World's most famed languages for example; C++, Python, Java, C#, PHP, Dart, Ruby among them [16]. The study of [7] proves it a challenge for students to learn the object-oriented concepts like Java, Python, or C++ because they are infused with intangible concepts like class, inheritance, and polymorphism. Other challenges that can be seen also include problem complexity, programming environment, and students' skills which can also hinder learning. For the better learning, such approaches in which interactivity or context awareness is utilized have been implemented like hands-on learning or those that use figures instead of normal programs [13]. [11] Suggest that following exposition to event handling and function call blocks, other complexities and abstraction should be introduced since the problem calls for a certain degree of abstract thinking, logical capabilities, and conventional programming skills. The procedural or structured codes fine for small tasks only, as debugging if the coding length is large and enhanced is difficult. The re-usability aspects of code incorporated in OOPs as well contribute to it being a clearer and less complicated coding style. However, with the employment of OOP coding techniques, some constraints have been realized including the issues of complexity control [10]. Examining the current and past essence of software engineering, three OO programming languages are Java, C++, and Python. Java has managed to entrench itself as the most popular OO programming language for many reasons and these are; the network centric independent java platform, Powerful set of java applications program interface (Java APIs).

However, Java lacks versatility when it comes to the implementation of the multiple inheritance. [3]. [15]: C++ is another programming language which is used commonly and is umpire considered the most comprehensive one owing to its support to procedural, modular, data abstraction and object-oriented as well as generic programming. As stated in [9], Python is an open source robust and high-level object-oriented general purpose programming language developed by Guido van Rossum [6]. Its uses cuts across the web development, scientific and mathematical computing, and even graphical user interface on the desktop. It is a basic language; free, can be used on any operating system, can be added onto and incorporated, works through translation, and has large libraries to help accomplish everyday operations. Python support same as C++: single and multiple heirs. Object Oriented Programming is a great revolutionary improvement in the programming construction. Institutions of some object – oriented languages were constructed by numerous years of experience within software development that tends to motivate construction of things such as closed methods, modules and concealed data. [8]. OOP intervenes with the at least three software engineering goals; namely reusability, extensibility and flexibility/polymorphism [2].

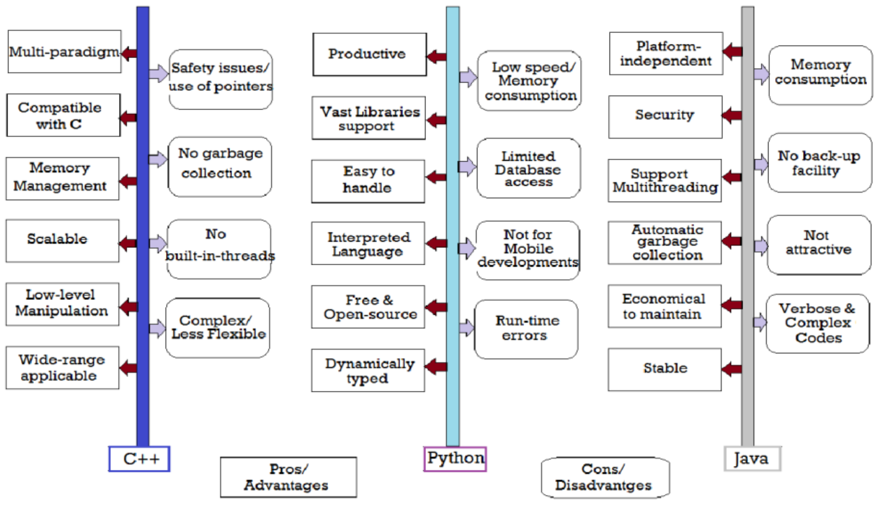

Fig 1: Comparisons of C++, Python and Java programming languages and their advantages and disadvantages [14].

In this context the paper attempts to understand the Object Oriented Programming Concept and some of the main issues in the implementation of the Object Oriented Programming Languages.

(i) **Encapsulation:** In OOPLs, Encapsulation is the ability to group data together with the operations that are performed on that data and form a class. It assists in restricting the exposure of state information in an object while a number of necessary methods are provided, which makes it effective in maintaining data purity and security.

Some of the major complexities in encapsulation within OOPLs include: Some of the major complexities in encapsulation within OOPLs include:

Access Control: Concern of controlling the accessibility of the attributes and methods within the class' code in order to avoid unlimited access and/or modification. -This includes grasping concepts such as access specifiers which are public, private, and the protected.
-Getter and Setter Methods: Allowing controlled access to the class attributes by using the concepts of the getter and setter method. This complicates the code base with one more layer and that needs to be designed with how we are going to use the functionality in mind.
-Design Patterns: When trying to employ design patterns such as Singleton, Factory, Observer, as well as the object encapsulation, it may be rather difficult. Such patterns actually involve competencies when it comes to revelation or the better part which is the concealment of aspects of the class.
-Serialization and Persistence: There are challenges in making certain that the data encapsulated therein can be serializable and that they could be persisted without the level of encapsulation being violated. This includes coordination of serialization of private data and methods.
-Testing: Actually, it can be noted that testing of encapsulated classes is relatively more difficult than testing of non-encapsulated classes. This means that test cases must be developed to check that values passed between encapsulated data and methods are correct for that specific type of environment.

(ii) **Inheritance:** is one of the key features in most Object-Oriented Programming Languages where one class (derived class or sub class) can take attributes and methods from another class (base class or super class). Despite the advantages of inheritance including code reusability and extensibility it comes with some form of complications. Here are some of the major complexities associated with inheritance in OOPLs: Here are some of the major complexities associated with inheritance in OOPLs:
-Inheritance Hierarchies: Since the classes that are being derived from a common base class increases, the management of the numbers of hierarchy created also increases. Another issue of deep hierarchies is the diamond problem where a class could be inherited through two different paths from the same superclass.
-Method Overriding: When a child class gives an individual implementation of the method that exists in the parent class, the process is referred to as method overriding. This can cause slight bugs if not well handled especially when the original method being overridden has intricate operations or impacts.
-Access Control: Inheritance also affects and is affected by the access specifiers which include public, private and protected in unexpected ways, Members of subclasses are inherited from super classes according to their level of access control, which may cause certain problems concerning visibility or encapsulation.
-Multiple Inheritance: A few languages allow for multiple inheritance that is when a subclass can be created inheriting from two or more classes. Though it's strong, MI has its drawbacks that can become fatal: the presence of ambiguity; the diamond problem, which complicates code readability and maintainability.
-Fragility: Modifications done to the superclass will cause impacts on the subclass, which may lead to alterations of numerous subclasses within the code structure. This can lead to the formation of a rather fragile code and, in consequence, can result in considerable problems in code maintenance.
-Testing: It is worth to mention that in framework of unit testing, inheritance could potentially cause significant problems since the inherited code, present in the sub classes, also has to be tested. This translates into a consideration on how best to conduct and organize tests in order to obtain adequate coverage.

(iii) **Polymorphism:** The term comes from the two words poly that means many and morph that means form. Polymorphism is a concept which gives the meaning to the word poly, which means many, as in an object can take a many forms. Thus, OOP prevents the growth of constraints dependency by means of polymorphism. [12].

Actually polymorphism is a very important concept used in Object-Oriented Programming Languages (OOPLs) which allows objects of different classes to be used as objects of a common superclass. This allows the ''one interface'' to refer to distinct forms or data types at its base. Like any powerful and flexible concept, polymorphism has both its benefits and drawbacks; it's the extensibility issues that are complicated in itself. Here are some of the major complexities associated with polymorphism in OOPLs: Here are some of the major complexities associated with polymorphism in OOPLs:

-Method Overriding: Polymorphism sometimes requires method overriding in which the subclass gives a concrete implementation for a method that is already implemented in the superclass. This begins to give rise to degenerate bugs if not carefully handled, particularly when the overridden method possesses a complexity or taxonomic effects.

Dynamic Binding: Polymorphism is based on the Dynamic Binding which is also called Late Binding or Runtime Binding the involved method is determined at the runtime based on the actual type of the object
- It is costly as it introduces extra use time and may narrow the range of results returned, thus potentially slowing down the system.
-Interface and Abstract Classes: Subtyping or implemented/reduced typing, in which polymorphism can be realized through interfaces or abstract classes that establish a contract to be fulfilled by subclasses. Operations with interfaces and abstract classes, particularly in large-scale projects that involve various implementation instances, may cause some difficulties due to the necessity for proper planning.
-Covariant Return Types: Covariant return types enable the overridden methods to return a subtype of the class declared in the superclass. Although it aids polymorphism, this can as well lead to confusion and slight mistakes provided it is not correctly used.
-Method Overloading: Method overloading enables different methods to be defined with the same name but different parameter list either in the same class or in different classes. While this increases polymorphism, it may always be a source of confusion and vagueness if not well applied.

(iv) **Abstraction:** is one of the significant concepts in Object-Oriented Programming Languages (OOPLs) which helps the programmer to represent the actual world object into classes and manage to show its essential properties while other non-primary properties' implementation can be handled internally. Thus, abstraction has its advantages, at the same time, it is not devoid of some complications. Below are some of the major complexities associated with abstraction in OOPLs: Below are some of the major complexities associated with abstraction in OOPLs:
-Choosing the Right Level of Abstraction: When it comes to the issue of abstraction the main problem is often what to abstract and at what level. Misuse of abstractions may cause high complexity of code that is hard to understand, on the other hand, little use of abstractions may result in high-coupled implementation dependent code.
-Design Patterns and Abstraction: Abstraction is frequently used in design patterns to offer up solutions to recurring design issues. However, grasping design patterns sensibility and learning the best ways to use them effectively needs fundamentals of abstractions, and how to achieve them.
-Understanding and Using APIs: It is commonly seen in Application Programming Interfaces (APIs) where one interface may be built on top of another in order to offer the consumer a simplified view of a vast set of capabilities. Fortunately, how to use and extend APIs and the principles of abstraction boundary within them are different and not obvious questions that need to be solved with precision and creativity.
-Testing Abstractions: It is usually not easy to test abstract classes and interfaces, because they frequently do not have implementations. Sometimes unit testing may require the creation of mock or stub objects in order to drive them and this also complicates the testing process.

III. Methodology

Object-oriented programming (OOP) has made a very high impact in modern programming. It has also provided the basic stand for most Artificial intelligent projects. There are features of OOP languages which allow programmers to develop complex and robust applications. However, for many programmers, these features can be challenging to understand. In this paper, we present a methodology for demystifying OOP features in popular OOPLs. The methodology we propose consists of the use of survey questionnaire tailored to extract information from various level of programmers on the complexities they encounter while implementing these features in any or all of the popular OOPL (Java, C#, C++ and Python) Our survey questionnaire was aimed to answer our research question and as well address our objective in this paper.

Research Question: what are the common challenges programmers encounter while working with Object -oriented Programming languages (OOPLs)?

Objective: Properly identify most challenging OOP features in popular OOPLs and analyze those errors and misconceptions of OOP features in OOPLs.

A simple random sampling method was used to select over 80 programmers in the Computer Science field in some Nigerian universities. We narrowed to those in Masters, PhD, and Lecturers with both theoretical and practical knowledge of Object-oriented programming languages. With our sample size, we believe it is enough to give us the confidence level of 95% with an error margin of 5%. Our survey questionnaire has fifteen (15) questions, which included close-ended and open-ended questions. The questions covered areas of interest, aiding us to understand perceived difficulties in implementing OOP features in OOPLs and the common errors with the misconceptions therein. This survey was conducted online and several weeks were allowed for our respondents to return the questionnaire. Seventy respondents completed and returned via email and other social platforms. It was based on our respondent answers, bearing in mind potential response bias that we subject our data for analysis. In our data analysis, we used descriptive statistics, inferential statistics, and content analysis. This provided insight to the most challenging features of OOP while developing software and the areas of confusion or complexities among programmers of different levels. Through our analysis we could understand the effectiveness of demystifying OOP features in improving programmer productivity and reducing errors.

IV. Result And Discussions

The problem that necessitated this study has to do with many of the complexities surrounding the implementation of OOP features in OOPLs. To get an insight into the nature and impact of the challenges faced by developers using the most common OOPLs as a case study, survey of 80 programmers with Computer Science background from various universities in Nigeria, 87.5% of our respondent completed and returned their questionnaires. The research kicked off seeking for answers to some troubling questions.

In this paper some key metrics were used to evaluate the results of analysis of survey questionnaires on demystifying Object -oriented programming features in popular Object-Oriented programming languages (OOPLs). They are as follows;

i. Response rate i.e the percentage of the survey questionnaire that were returned and completed.

ii. Demographics. We used only the level of programming skill not age or gender e.t.c

iii. Understanding of OOP concepts. There were questions in our survey that tested our respondent understanding of OOP concepts.

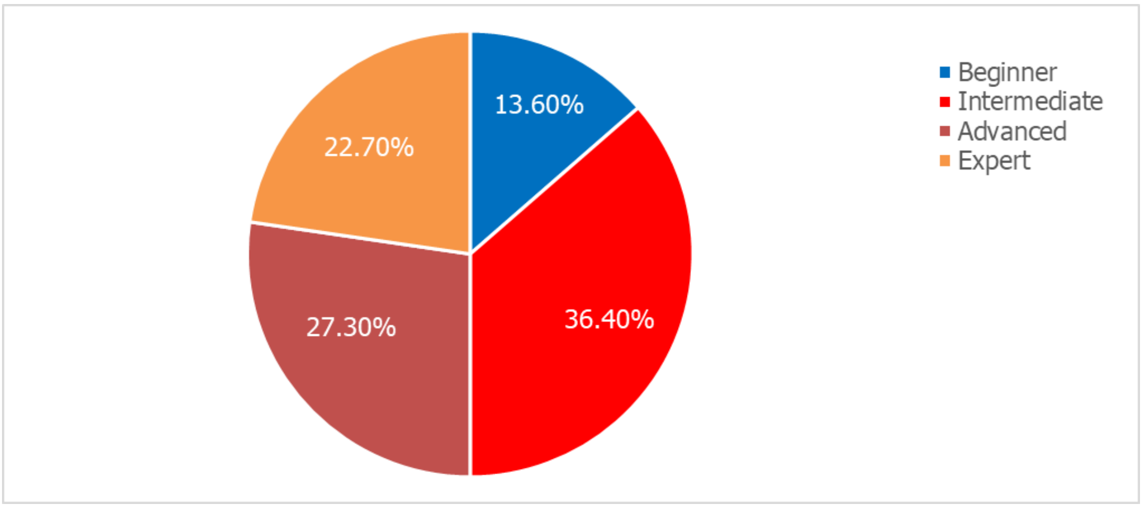

Fig 2: Level of programming skills of respondents.

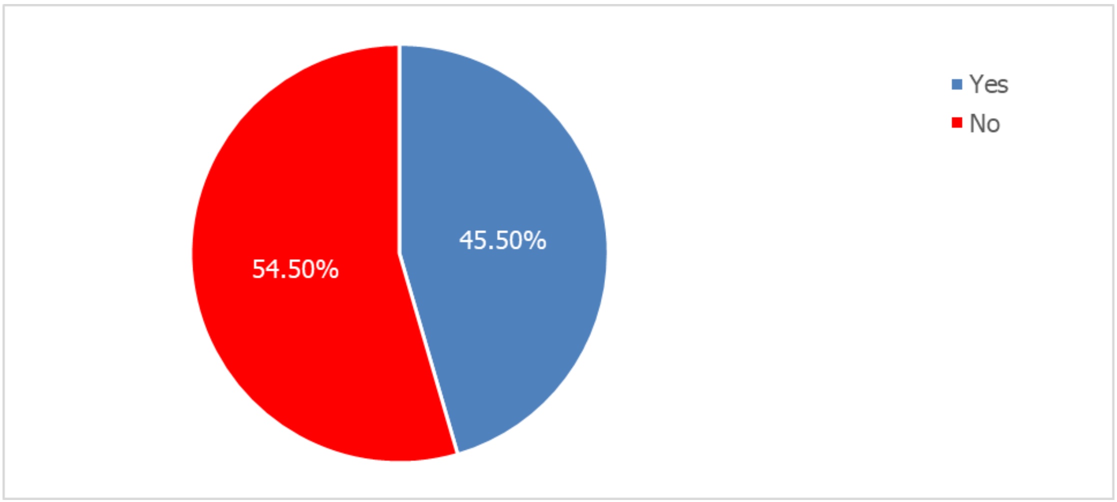

Fig 3: Respondent who believe there are similarities and those who believe there are no similarities when implementing OOP features across popular OOPLs.

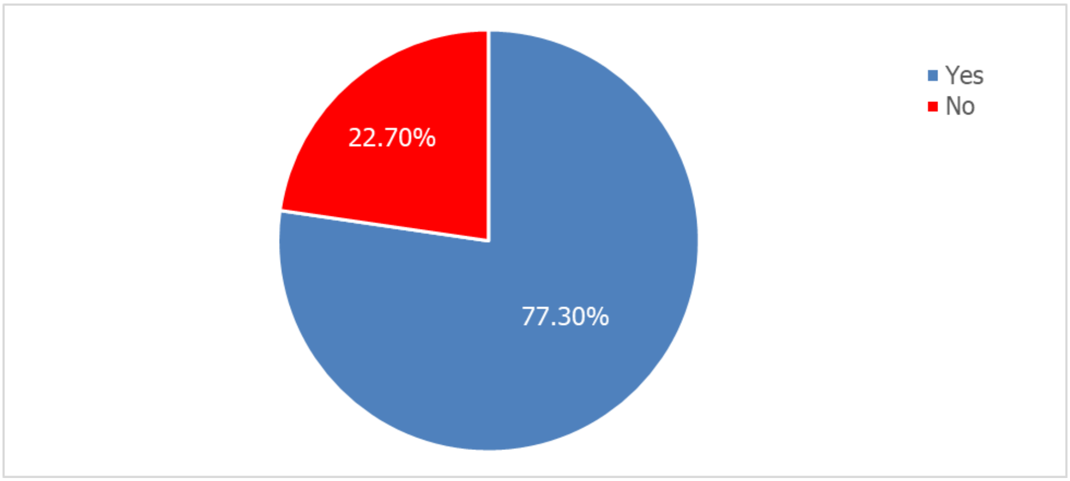

Fig 4: Respondent who believe there are differences and those who believe there are no differences when implementing OOP features across popular OOPLs.

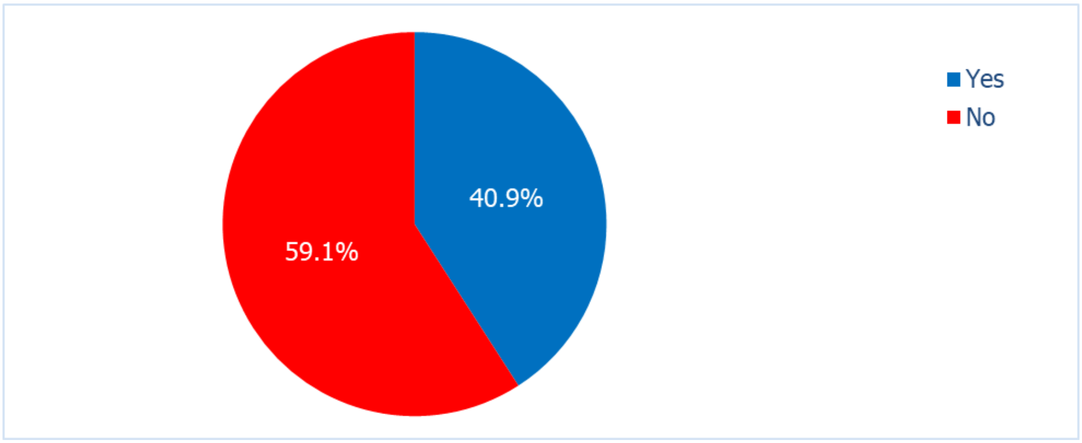

Fig. 5: Respondents requiring assistance and those who do not.

Figure 2 depicts the level of programming skills of our respondents, we concentrated from the level of intermediate to advance in our analysis to determine the common challenges encountered while working with OOP features in OOPLs and identify the most challenging OOP features, common errors and misconceptions

Figure 3, represents the percentage of those who believe there are similarities and those who believe there are no similarities when implementing OOP features in our popular OOPLs.

Figure 4, shows the percentage of those who believe there are difference and those who do not agree there are differences while implementing OOP features in those popular OOPLS.

Figure 5, represented the percentage of those that need help and those who do not need help implementing OOP features in OOPLs.

Our findings from this analysis are as follows;

40.9% needs assistance in implementing OOP features in those popular OOPLs while 59.1% claim they do not need any assistance.

1. In figure 3, 54.5% affirms that there are similarities when implementing those feature (Encapsulation, abstraction, inheritance and polymorphism) while 45.5% claims the contrary.
2. In figure 4. 77.3% agrees to the differences while 22.7% believes there are no differences while implementing the feature in those OOPLs.

These responses gave us the insight on various misconceptions and understanding among programmers with different skill levels. Some of the respondents that claim they do not need help actually need some help based on their positions in figure 3 and 4.

Our research proves that there are actually similarities and obvious differences when implementing OOP features in those mentioned popular OOPLs. This survey has given us the ground to understand some of these misconceptions and error that culminate to the various degree of complexities programmers in Nigeria universities encounter while working with those popular OOPLs. It should be recognized that all these popular OOPLs implement these common features (encapsulation, abstraction, inheritance and polymorphism) though some has different ways of implementing some of these features. All of these OOPLs supports classes and objects, Constructors and Destructors. We also noted that there are differences in syntax, data type and memory management. It is also noticed that one of them supports multiple inheritance while the rest three do not. They all supports access modifier but implements it differently. It is observed that understanding these similarities and differences will significantly reduce the complexities programmers experience while implementing OOP features in OOPLs.

V. Conclusion.

Our survey results indicate that a majority of software developers in Nigeria universities with computer Science background do not have a good understanding of OOP concepts in popular OOPLs. However, small portions of developers do but may still need further education or clarification on these concepts. This suggests that there is a need for training and education for software developers to improve their understanding of OOP features. The findings can be used to inform the development of training materials and courses which will focus on OOP concepts for software developers with the goal of improving software development practices and enhancing the quality of software products.

References

[1] Adetiba, E., Adeyemi-Kayode, T. M., Akinrinmade, A. A., Moninuola, F. S., Akintade, O. O., Badejo, J. A., Obiyemi, O. O., Thakur, S. & Abayomi, A. (2021). Evolution Of Artificial Intelligence Programming Languages - A Systematic Literature Review. Journal Of Computer Science, 17(11), 1157-1171. Https://Doi.Org/10.3844/Jcssp.2021.1157.1171

[2] Asha R. M., Kavana M. D., Parvathy S. J., Shreelakshmi C M., (2017) Object-Oriented Programming And Its Concepts. International Journal For Scientific Research & Development| 5(9) 840-842.

[3] Fawzi A., & Amjad M., A., (2017) Comparative Study On The Effect Of Multiple Inheritance Mechanism In Java, C++, And Python On Complexity And Reusability Of Code. International Journal Of Advanced Computer Science And Applications 8(6) 109-116.

[4] Goel, B. M.; Bhatia, P. K Analysis Of Reusability Of Object-Oriented System Using Ck Metrics. International Journal Of Computer Applications, 2012, Vol. 60, No. 10, Pp 32-36.

[5] Gupta A.; Dashore P. An Approach To Analyse Software Reusability Of Object Oriented Code. International Journal Of Research In Science & Engineering, 2017, Volume 3, Issue 1.

[6] Hemmendinger D.,"Object-Oriented Programming | Object-Oriented Programming | Classes, Encapsulation, Polymorphism", The Editors Of Encyclopedia Britannica(Britannica) (2008).

[7] J. Szydlowska, F. Miernik, M. S. Ignasiak, And J. Swacha, "Python Programming Topics That Pose A Challenge For Students," In Third International Computer Programming Education Conference (Icpec), 2022.

[8] Kafura. Object-Oriented Programming And Software Engineering. 1996 Available From: Http://People.Cs.Vt.Edu/Kafura/Cs2704/Oop.Swe.Html.

[9] Lutz, M. Learning Python, 5th Edition. Published By O"Reilly Media, Inc. (June), Ca, Usa, 2013.

[10] Massimiliano Dessì, (2009) "Spring 2.5 Aspect Oriented Programming", Packt Publication, Pages 332, Isbn: 9781847194022.

[11] M. Csikszentmihalyi, Flow: The Psychology Of Optimal Experience. Harper And Row New York, 1990.

[12] Nzerue-Kenneth P.E., Onu F.U., Denis A.U., Igwe J.S., Ogbu N.H. (2023), Detailed Study Of The Object-Oriented Programming (Oop) Features In Python. British Journal Of Computer, Networking And Information Technology 6(1), 83-93. Doi: 10.52589/Bjcnit-Facsojao

[13] Qais A. B., Nazatul A A., Noraidah S., Noorazean M.A., Analysis Of The Learning Object-Oriented Programming Factors. International Journal Of Electrical And Computer Engineering. 13(5) 5599-5606.

[14] Singh, N; Chouhan, S; Verma, K (2021): Object Oriented Programming: Concepts, Limitations And Application Trends. Techrxiv. Preprint. Https://Doi.Org/10.36227/Techrxiv.16677259.V1

[15] Stroustrup, B. The C++ Programming Language, Fourth Edition. Addison-Wesley, 2013.

[16] Vishal V. M., Yash N., (2019) Review On Concepts Related To Object Oriented Programming System, Iconic Research And Engineering Journals.3(6) 56-58.

[17] Walelign Tewabe, The Object Oriented Software Development For Artificial Intelligence