Available Online at www.ijcsmc.com

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X

IMPACT FACTOR: 6.199

IJCSMC, Vol. 9, Issue. 1, January 2020, pg.65 – 74

A REVIEW OF ASPECT ORIENTED PROGRAMMING FOR ENHANCED PROGRAM MODULARITY

Nnaa, Sunday Barikui[1]; Ojekudo, Nathaniel Akpofure[2]
Department of Computer Science, Ignatius Ajuru University of Education, Rumuolumeni, Port Harcourt, Rivers State, Nigeria
+2348032394805, nnaasunday76@yahoo.com [1], +2348037581323, nathojekudo@gmail.com [2]

Abstract: Software development process has evolved from the traditional software development paradigm with the attendant difficulties to a more intuitive approach. With the various improvement injected into the software program development landscape by the introduction of object oriented programming in terms of the modularization process, there abound a huge limitation due to the presence of cross-cutting concerns that have undermined the principle of abstraction with the attendant code scattering and tangling but with the inception of Aspect oriented programming and its complementary role to traditional OOP, there has been an improvement in the effective management of these concerns through the use of Aspect which focuses on specific cross-cutting functionality by unburdening its core classes, improving modularity and program efficiency. This paper reviews this two software development paradigm with a view to ascertain the common ground and visible difference amongst them and how AOP is changing the software development process.

Keywords: Modularity, AOP, OOP, Aspect, Cross-cutting concerns

Introduction

From the basic perceptions of programming does not only intend to automate processes, but also focuses on process optimization and efficiency. The different programming paradigm over time has induced some form of enhancement or the other to the software development process but there are still rooms for improvement as the process of use and development has thrown up new challenges and opportunities for further improvement. Several approach abound in the program development space which over time has influenced the way software programs are developed and deployed, these programming paradigm includes procedural, structural, functional, declarative, modular, object oriented programming and more [1].

Software development paradigm devices a distinct methodology to program development that is unique in itself. There has in recent time being a drastic paradigm shift from the traditional object oriented programming (OOP) paradigm to a more elaborate and inductive aspect oriented programming (AOP)approach based on some perceived limitation in terms of code tangling and scattering which down played the principle of abstraction inherent in the former [2].This paper is intended to review the basic concept in aspect oriented programming and its relationship with the core object oriented programming paradigm with respect to managing cross-cutting concerns and modularity in program design and implementation.

Furthermore, we shall consider the concepts of modularity, object oriented and aspect oriented programming respectively with the view ascertain the relationships amongst these programming paradigms.

Modularity in Programming

As the name implies, modular programming entails the breaking down of programming tasks into modules or constituent parts that runs independently to achieve set goals, modularity facilitates the grouping of some lines of code into smaller unit that can be incorporated in the main program. Modular programmings employs the divide and conquer approach by breaking and dividing the program components into manageable bits making it reusable and easy whereby existing program blocks can be used to create new ones. It also enables for ease in the separation of the functionalities of a program into independent blocks to extent that each of these program blocks or module could be executed independently with full functionality in the desired target area, called and reused over and again without the loss of its functionality[1, 3].It is a process that reduces program development time and allows for the independent development of each modules or components without altering other components or modules in the program[4]. Software modularity is one of the key characteristic for the management of program complexities of as it enhances the reusability and maintainability of systems and mostly deployed in object oriented programming paradigm through the use of methods, classes and others [5].

Object Oriented Programming (OOP)

With the rapidly growing global population and the resultant pressure and demand on businesses to meet their growing customer’s needs, the place of a multi-dimensional software development approach to automate this growing business challenges cannot be overemphasized. There is also an increase in the algorithmic complexities in conventional software development paradigm making it inefficient for managing massive business computing and automation. To overcome this challenge, object oriented programming approach emerged as a key programming paradigm to reduce the inherent algorithmic complexities and develop software programs that are easy, reliable and maintainable [6]. Object oriented programming paradigm is an approach that simplifies software development processes by modelling the solution to a problem in to a collection of collaborating objects sending massages one to another[7, 3]. Das[8]further opines that the underpinning concepts of OOP which includes data abstraction, hiding, encapsulation inheritance, polymorphism, encapsulation and modularity was aimed at eliminating some of the challenges in procedural programming to create an evolutionary programming paradigm by providing a distinct framework for the development of innovative and flexible software solutions [9]. Here objects are the basic runtime entities whose attributes and functions are compressed in a class. Furthermore, functionalities are encapsulated in objects with the features of data and functions in a single unit[10]. From the foregoing Chandel[11], identified encapsulation, polymorphism, abstraction and inheritance as the major concepts which he also christened the pillars of object oriented programming as all other principles of OOP are hinged directly to them.

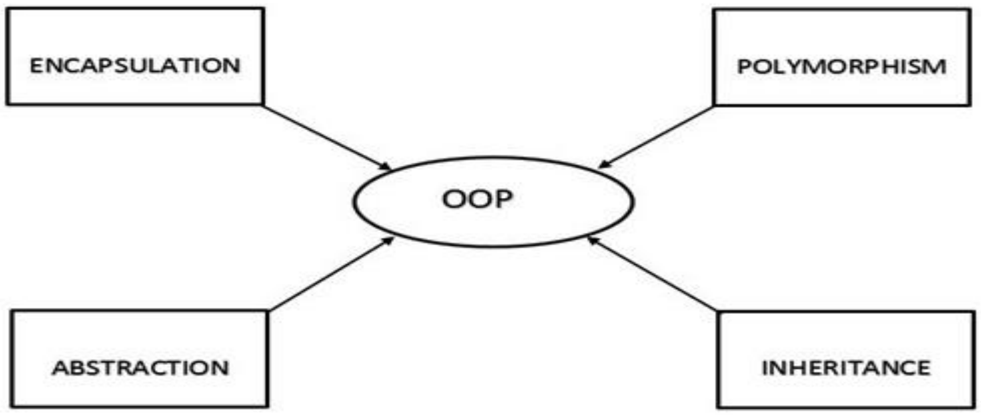
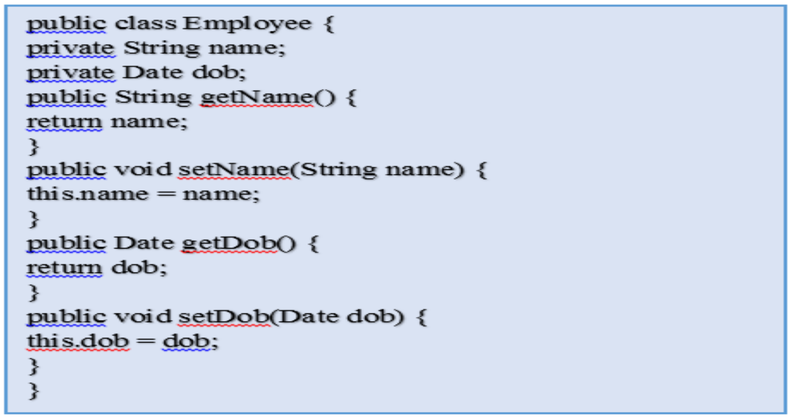


Figure 1: Pillars of Object Oriented Programming (OOP) (Source: Chandel, 2018)

Encapsulation

Encapsulation in OOP is the process of compressing data and other membership functions to form an object. It involves the hiding of implementable data by placing restricted access to methods keeping instance variable private and accessor method or getter public[1, 11]. The code snippet below show encapsulation or compression of name and dob attribute of the Employee class.



```
public class Employee {
private String name;
private Date dob;
public String getName() {
return name;
}
public void setName(String name) {
this.name = name;
}
public Date getDob() {
return dob;
}
public void setDob(Date dob) {
this.dob = dob;
}
}
```

Polymorphism

Polymorphism is the concepts that allow the use of the same function for different purposes. It allows for the passing of the object of a resultant class a given function that accepts the reference to base class. Polymorphism is the mechanism that enhances OOP by ensuring that objects of a resulting or derived class are treated as if they are of the base class [12].

```
publicvoid print(Collection<String>
collection) {
for (String s : collection) {
System.out.println("s = " + s);
}
}
```

Abstraction

Data abstraction is the representation of data in which the implementation details are concealed or hidden. Cline and Girou [12] posits that it is the simplified view of an object that informs the user of the basis of an object without detailed implementation breakdown expressing the intent of the given class rather than its implementation. Abstractions are formed by well-defined user interface designs that shields the users from the actual implementation framework that drives the program.

Inheritance

This is the process where a class inherits the functionalities of another class also known as the parent class. Here the child class either inherits all or some of the attribute such as member function and variables of the parent class[1]. Inheritance encourages the definition of new classes which the functionalities of the class enhancing the process of code reusability where in a derived class codes could be reused from an existing super class and further extended[13].

Limitations of OOP paradigm

Despite the innovation and enhancement that object oriented programming paradigm brought to the software development landscape there abound some notable limitations that undermines is performance and productivity. Since software systems are conceptually complex in nature and an increase in their complexity indicates the possibility for their failure [13]. Some of these limitations in applications of OOP iscross-cutting concerns which includescode scattering and tangling.

Cross-cutting concerns

Conceptually, a concerns refers to the ability to identify, combine and manipulate related pieces or components of a computer program. They are the design related matters that depicts the requirement specification and information that affects the program's code[10]. Therefore cross-cutting concerns are a concern that cuts across a number of program components and affects their functionalities cutting across other concerns too. These are independent entities that transversally cross other functionalities of software program. The most frequently observed cross-cutting concerns includes; security, logging, transactions management, caching, performance checking, concurrency control, and exception management[13, 10].From the foregoing, one critical cross-cutting concern that affects others concerns in a software program that is difficult to completely isolate is logging[14].

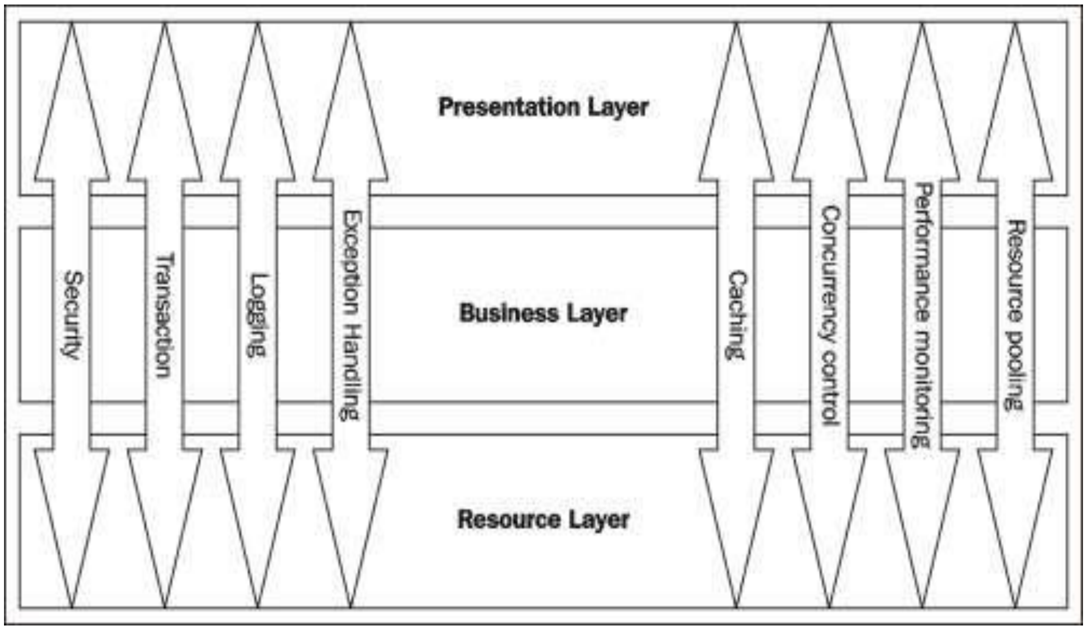


Figure 2. Graphical illustration of Crosscutting Concern (Source: Dessi, 2009)

Figure 2 illustrates the effect of cross cutting concern in a software system as it traverses different concerns in the program. This indicates that the traditional object oriented paradigm's ability to modularize the program into smaller components are here undermined by these concerns as they could result to a possible failure of the system being developed. Felix and Ortin[5] further asserts that these concerns cannot be directly modularized in the object oriented programming paradigm as it does not have enough expressiveness to implement them as independent modules.

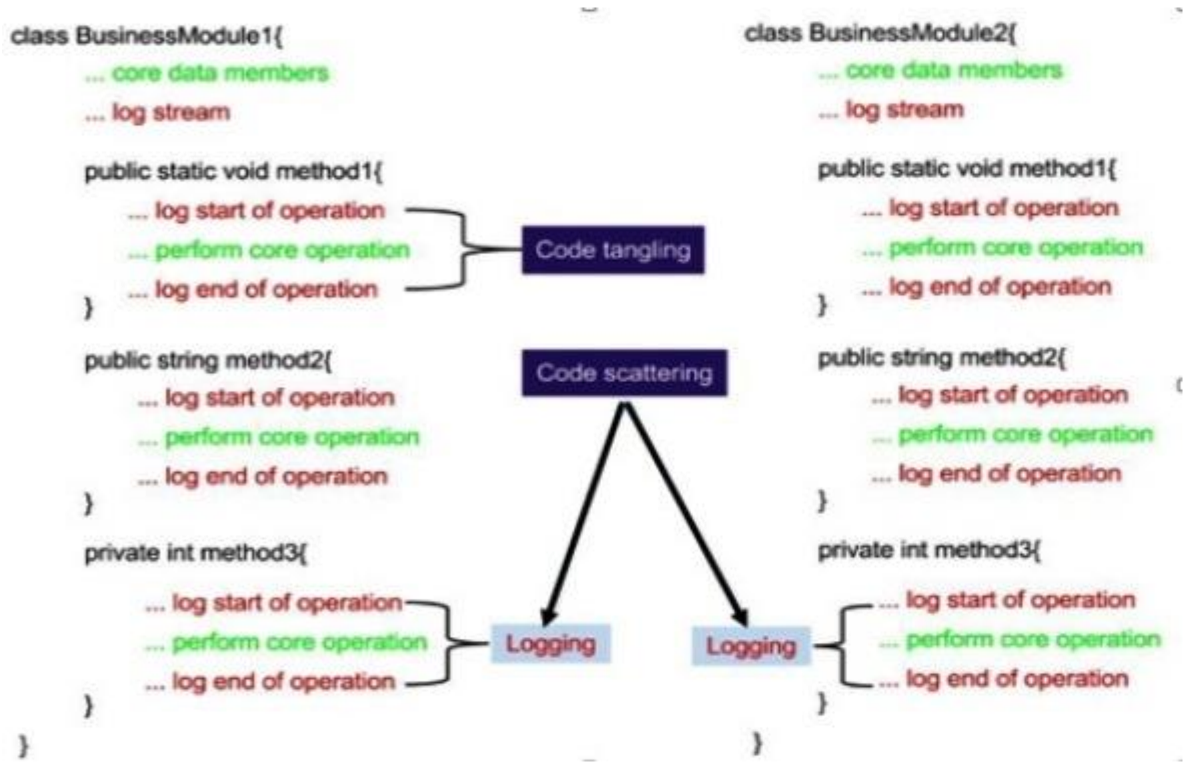
Code Scattering

Code scattering is the process whereby a code performing a given task or function is scattered across different modules in the program. Here the functionalities of the said code is not concentrated on a particular module but are either block of duplicated codes so that the same code appears in different modules of the program or block of complementary codes where different modules implements the corresponding unit of the same concern[13].

Code Tangling

Code tangling occurs when a given piece of code is made to perform several or multiple function. Dessi[13], asserts that it is a situation where a given program module contains in itself components that are required for the implementation of other concerns outside it, whereby such module is made to manage several concerns at the same time.

In Figure 3, shows the scattering of codes for logging process scattered across two modules in a program and the start and end log code in method 2 tangled. The implication of this in object oriented program development is that it will complicate the development and create difficulties in the maintenance process[14]. Furthermore, it could cause the duplicate codes and functionalities not to be clear and with the implementation of further requirements, there may arise strong module coupling in the implementation process[13]. Other limitations include; poor quality software, inability to reuse codes and difficulties in tracing faults due to code scattering.



```
class BusinessModule1{
    ... core data members
    ... log stream

    public static void method1{
        ... log start of operation
        ... perform core operation
        ... log end of operation
    }
    public string method2{
        ... log start of operation
        ... perform core operation
        ... log end of operation
    }
    private int method3{
        ... log start of operation
        ... perform core operation
        ... log end of operation
    }
}
```

```
class BusinessModule2{
    ... core data members
    ... log stream

    public static void method1{
        ... log start of operation
        ... perform core operation
        ... log end of operation
    }
    public string method2{
        ... log start of operation
        ... perform core operation
        ... log end of operation
    }
    private int method3{
        ... log start of operation
        ... perform core operation
        ... log end of operation
    }
}
```

Figure 3: Crosscutting Concern Problems (Source: Program Creek, 2011)

Aspect Oriented Programming (AOP)

Due to the difficulties in modularizing software development process and the attendant concerns in conventional software development paradigm, there has been a shift in the software development community towards aspect oriented programming (AOP), a new program development approach that manages the separation of concerns and enhance program modularity. This move from OOP to AOP was predicated on the inevitability of the inability of the former to effectively manage cross-cutting concerns. Unfortunately, the presence of code tangling and scattering has undermined the principle of abstraction inherent in OOP and these inbuilt challenges are problems that is being resolved by the concepts of AOP[2].

AOP is a program development solution designed to address and manage the problems and defects in software modularization that is predominant in conventional approaches by focusing on the identification, specification and representation of cross-cutting concerns and their modularization in each phase of the program development process. It is not made to replace the former by it is a complementary approach to the former, in other words it facilitated and enhances the operation of the traditional software development paradigms[15, 5].Biswal[10], further opines that aside from its complementary role, it facilitates program modularity by encapsulating cross-cutting concerns into an aspect which is a separate module in the program equivalent to a class in OOP, implementing methodologies such as advices and point cuts which then refactors crosscutting concerns into aspects.

The function of the aspet is to focus on a specific cross-cutting functionality thereby unburdening the core classes of these concerns. The final system is then composed by an Aspect weaver which combines the main classes and the cross-cutting concerns through a weavering process, thus creating an application that is easy to design, implement and manitain[16].

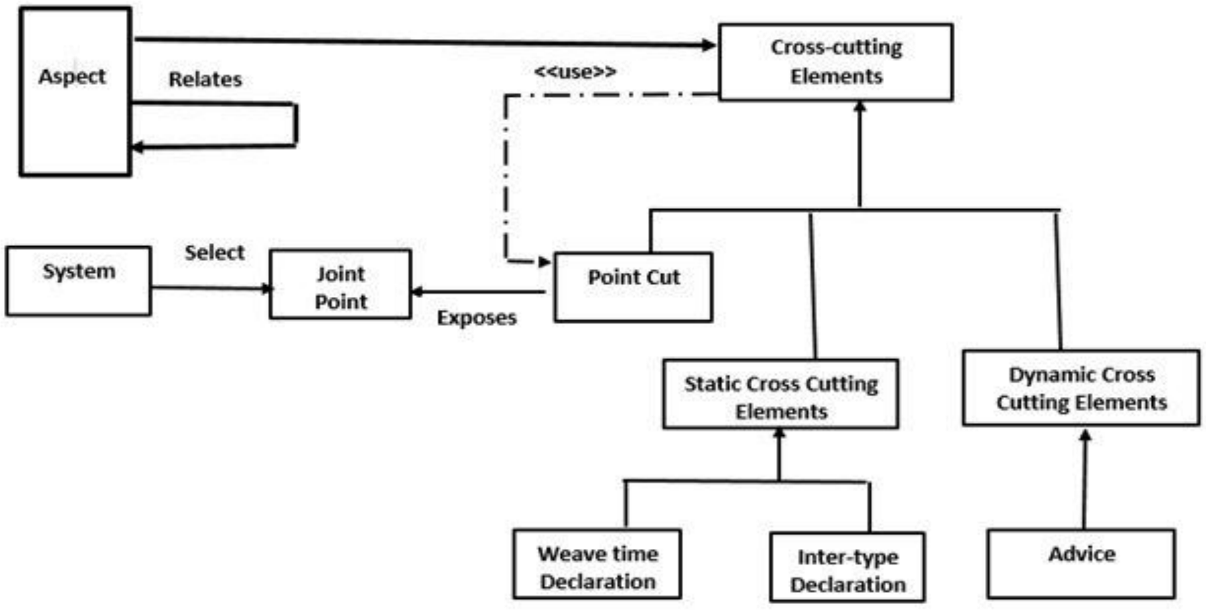


Figure 4: Generic Model of AOP Sustem (Source: Kumar et al, 2016)

The major advantage of AOP is that the use of codes are easy to read, maintain and not easily disposed to bugs, which in turn facilitate and speed up program development and boost the morale of development team[16]. By its construct AOP allows for the uncoupling of modules, modularization of cross-cutting concerns and the use of Aspects to remove cross-cutting concerns by the modules that use them [13].Gulia et al[17], also opines that comparative advantage of the AOP over the OOP can be traced to its response time as AOP shows reduced response time on implementation and increased throughput making program development much easier and reliable. Figure 4, show the generic model of the Aspect oriented programming (AOP) approach. Kumar et al[16]and Dessi[13]further posited the following as some of the core component of AOP;

a. **Aspect:**
   Aspect in AOP are akin toa class in OOP and contains the cross-cutting functionality of the program, handling the encapsulation of join points, point cuts, and advice allowing for comprehensive encapsulation of code associated to a specific concerns.
b. **Join point:**
   This is the application point of the aspect. These could be method calls, constructor invocations, exception handlers, or other points in the execution codes of a program where concerns will cross-cut the application.
c. The language type that matches the join points.

d. Advice

Advice are the action performed by an aspect at any given joint point. They are executed in three phases before, around and after also before any advice is executed, a point cut must be triggered.

e. Cut Point

A cut point is a construct that is used to express the join point selection used to group designators that uses the join point as a parameter. This parameter is further used to inform the aspect to match a public string join cut when it is part of a method call.

f. Weaving

Weaving is the process that is used to connect an object to an aspect to which an advice would be implemented or performed. Before an application execution, aspect can be statically and dynamically performed. AspectJ, an aspect oriented programming extension for Java likewise provides load-time aspect weaving at the instance a class is about to load into memory by a virtual machine. Aspect weaving takes instruction from the aspect oriented programming and produce the final result[13, 5].

Managing Cross-cutting Concerns, the AOP approach

From the foregoing we can say that AOP is specifically designed to manage crosscutting concerns at the system level and enables program or software developers to unmistakably separate cross-cutting concerns that would somewhat be entangled through an implementation[2]. Figures 5 and 6 shows the typical operations of object oriented and aspect oriented program paradigm and how the communications and interactions with modules are conducted. In the conventional OOP, classes communicates and corporates with each other through the process of calling methods and at the implementation phase of the individual class as illustrated in the figure 5 (class A, B and C) which results in tangling and scattering of codes in the system[13]. When an object in a given method as in figure 5, object A in method A calls or invokes another method B on object B these interactions are called back and redeployed because there is no other way out.

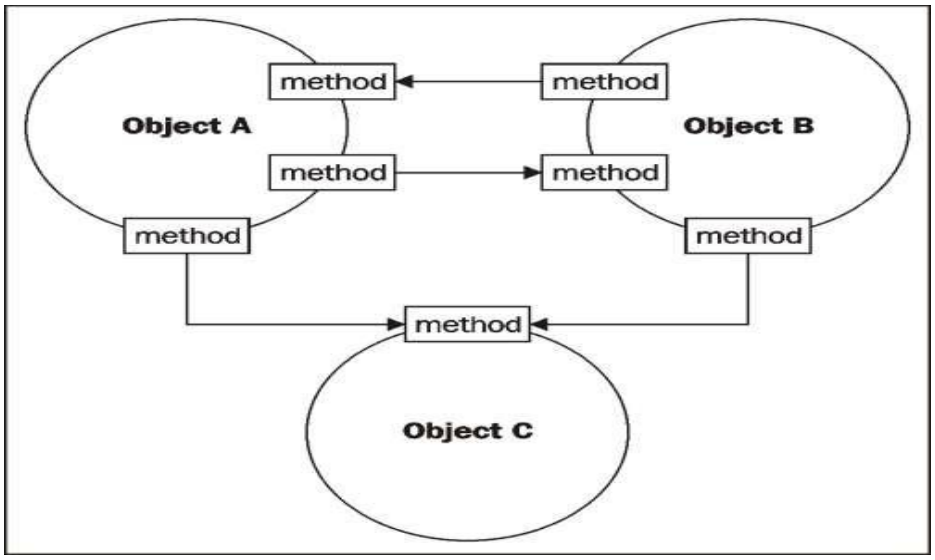


Figure 5 Object Interaction in OOP (Source: Dessi, 2009)

The AOP approach shown in figure 6, reveals that the cross-cutting functionalities are retrieved from the oriented approach implementation and used as an advice because they as implemented in the process where they are required to be carried out in the point-cut and target object.

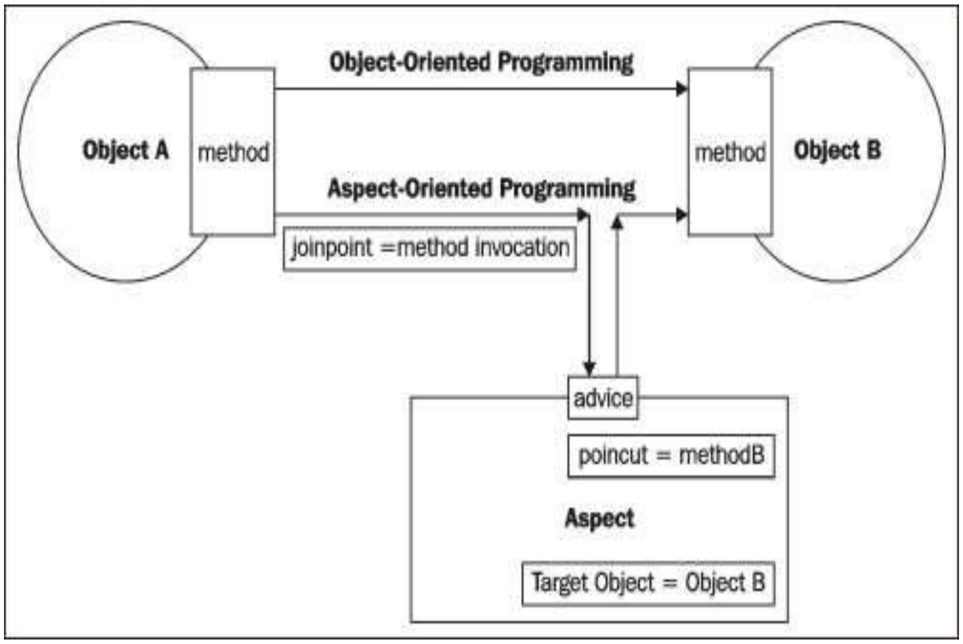

Object-Oriented Programming
Object A
method
method
Object B
Aspect-Oriented Programming
joinpoint =method invocation
advice
poincut = methodB
Aspect
Target Object = Object B

Figure 6: Object Interaction in AOP (Source: Dessi, 2009)

Therefore, figure 6 takes into cognizance that the join point invokes or calls the method B and the aspect executes the cross-cutting concern within the advice when method B is called or invoked on the object B this is also known as Before Advice as earlier stated in the different types of advice[13]. It is evident that AOP promotes separation of concerns as illustrated in figure 6 providing for cleaner assignment of codes and higher modularization making it easy for the program to evolve collecting scattered codes and other concerns [2].

Conclusion

Object oriented programming paradigm brought about improvement in programming and more insight into software modularization but not without the limitations of cross-cutting concerns like code scattering and code tangling, a programming defect that was has been addressed by the introduction of the complementary aspect oriented programming paradigm. AOP promotes the separation of concerns, effectively manage cross-cutting concerns by the use of aspects and eliminates code tangling and code scattering thereby greatly improving the quality of software developed by reducing implementation response time and facilitate throughput. The emergence of AOP paradigm has greatly improved the software modularity which has drastically reduced the difficulties experienced in software evolution. There is room for improvement as this review captures the key factors that surround the AOP revolution and its complementary role to OOP in the software development landscape.

REFERENCES

[1] N. Ojekudo, Computer Programming Bridge, Port Harcourt, Nigeria: Emmanest Ventures & Data Communication, 2019.

[2] Suboti, S., Bishop, J and Gruner, S, "Aspect-oriented programming for a distributed framework," *South African Computer Journal*, pp. 1-9, 2006.

[3] Deitel, P.J and Deitel, H.M, JAVA: How to Program, 7 ed., New Jersey: Pearson: Prentice Hill, 2007.

[4] Barbosa, F.S and Aguiar, A, "Modeling Crosscutting Concerns with Roles," in *The Seventh International Conference on Software Engineering Advances (ICSEA)*, IARIA, 2012.

[5] Felix, J and Ortin, F, "Aspect-Oriented Programming to Improve Modularity of Object-Oriented Applications," *Journal of Sotfware*, vol. 9, no. 9, pp. 2454-2460, 2014.

[6] D. Das, "Basic Concept of Object Oriented Programming Language," CSE Tutor, 2018. [Online]. Available: https://www.csetutor.com/basic-concept-of-object-oriented-programming/. [Accessed 12 November 2019].

[7] K. Richard, "Chapter 3: What is Object-Oriented Programming?," Medium, 2017. [Online]. Available: https://medium.com/learn-how-to-program/chapter-3-what-is-object-oriented-programming-d0a6ec0a7615. [Accessed 12 November 2019].

[8] D. Das, "Advantages and Disavantages of Object Oriented programming," CSE Tutors, 2018. [Online]. Available: https://www.csetutor.com/advantages-and-disadvantages-of-object-oriented-programming-language/. [Accessed 12 November 2019].

[9] Gulia, P., Khari, M and Patel, S, "Metrics Analysis in Object Oriented and Aspect Oriented Programming," *Recent Patents on Engineering*, vol. 13, no. 2, 2019.

[10] K. S. Biswal, "A Combined Approach for Identifying Crosscutting Concerns in an Object Oriented System," NIT, Rourkela, Rourkela, India, 2015.

[11] M. Chandel, "What are four basic principles of Object Oriented Programming?," Medium, 2018. [Online]. Available: https://medium.com/@cancerian0684/what-are-four-basic-principles-of-object-oriented-programming-645af8b43727. [Accessed 17 November 2019].

[12] Cline, M and Girou, M, C++ FAQ, Boston: Addison-Wesley, 1999.

[13] M. Dessi, Spring 2.5 Aspect Oriented Programming: From Technologies to Solutions, Packt Publishing Ltd, 2009.

[14] Bergman, I and Goransson, C, "An Introduction to Aspect-Oriented Programming," Authors & Karlstad University, Karlstad, 2004.

[15] AOSD, "Aspect-Oriented Software Development As Explained By Professionals," Aspect, 2017. [Online]. Available: http://aosd.net/aspect-oriented-software-development-explained-by-professionals/. [Accessed 12 December 2019].

[16] Kumar, A., Kumar, A and Iyyappan, M, "Applying Separation of Concern for Developing Softwares Using Aspect Oriented Programming Concepts," in *International Conference on Computational Modeling and Security*, Bangalore, India, 2016.

[17] Gulia, P., Khari, M and Patel, S, "Metrics Analysis in Object Oriented and Aspect Oriented Programming," *Recent Patents on Engineering*, vol. 13, no. 2, pp. 117-122, 2019.