International Journal of Innovative Technology and Exploring Engineering (IJITEE)
ISSN: 2278-3075 (Online), Volume-9 Issue-4, February 2020

Facets of Aspect Oriented Programming

Priyanka Sarode, R. N. Jugele

Abstract: ***AOP-Aspect Oriented Programming is a new programming paradigm for separating crosscutting concerns that are generally hard to do in object-oriented programming. As AOP has better ability to handle crosscutting concerns than Object-Oriented Programming (OOP,) it supports to write more modularized and more sustainable code. In addition, numerous publications discuss about the advantages of AOP design and implementation. However, for this new programming paradigm the work is in its early stages. The paper is all about the surveyed and reviewed several facets of AOP.***

Keywords: Aspect Oriented Programming, Object Oriented Programming, AOP, OOPs, tangled, crosscut.

I. INTRODUCTION

The fundamental objective of programming language is to create a program, which makes the computer a useful device. The bigger the program is extra challenging task to manage it. Computer Science has practiced an advancement in program writing languages as well as systems from the machine codes of the most basic computers through idea such as formula translation. Other basic programming paradigms are procedural, structured, functional programming. Each of these phases in programming technology has progressive developer's skill to reach clear separation of concerns at the source code level. At present, the prominent programming is Object-Oriented Programming (OOP)—the idea is all about the writing the code about objects. Object-Orientation (OO) reflected in the entire spectrum of current software development. Writing complex applications such as graphical user interfaces, operating systems and distributed applications possible with OOP. Success at developing simpler systems leads to greater complexity [1]. Object orientation is a dominant technology, but has solid limitations.

The term "Aspect-Oriented Programming" (AOP) [2] came into existence sometime between November of 1995 and May of 1996, at the Xerox Palo Alto Research Center (PARC). Neither procedural nor object-oriented programming techniques are adequate to collect all of the important design decisions that program must implement clearly.

Revised Manuscript Received on February 28, 2020.
***** Correspondence Author
Priyanka Sarode*, Inter Institutional Computer Centre; RTM Nagpur University, Nagpur, India. Email: priya.s1011@gmail.com
R. N. Jugele, Department of Computer Science, Shivaji Science College, Nagpur, India. Email: rn_jugele@yahoo.com

This forces the execution of those design decisions to be scattered throughout the code, resulting in "tangled" code that is excessively difficult to develop and maintain. Kiczales present an analysis of why certain design decisions have been so difficult to capture in actual code clearly. This code call the properties of design decisions to address aspects and gives the reason why it is so hard to capture. For that the team cross-cut the system basic functionality and present the basis for a new programming method, called AOP, that makes it possible to clearly express programs involving such aspects, including appropriate isolation, composition and reuse of the aspect code [2].

Correct definition of the software is not specified but acknowledged many properties of the software are good amongst which the essential ones are understandability, maintainability, reusability and evolvability. Modularization and properties strongly associated with each other, which provide the effect of the possibility to cleanly encapsulate concerns of the software in separate modules [3]. A module is a work assignment, whose role is to localize and hide design decisions. This principle is known as the Separation of Concerns (SoC) principle [4] [5].

Aspect-Orientation contracts by means of the SoC, the concerns that crosscut the modularity of basic programming mechanisms and it targets to minimize the code and to offer advanced structure. SoC is the most significant aspect in software engineering. It denotes to the ability to identify, encapsulate and manipulate only those parts of software that are relate to a particular concept, goal or purpose [6] [7].

This paper gives the several facets of Aspect Oriented Programming. OOP's limitations for modularization mentioned in section II. In next section, on which terminologies AOP is built and the concept of AOP is discussed. Section IV describes the properties of AOP and the benefits of AOP detailed in Section V. Lastly, in section VI the conclusion and future scope is underlined.

II. CROSSCUTTING CONCERNS, CODE TANGLING AND CODE SCATTERING

A concern is an exact requirement that must be addressed in order to fulfill the all goal lines of the given system. Software is a set of concerns. For e.g. a banking system, it consists of the following concerns:

- Customer and Account management
- Interest calculations
- Inter banking transactions
- ATM transactions
- Persistence of all entities
- Customer care, and so on.

With these system concerns, a software project needs to address process concerns, such as comprehensibility, maintainability, traceability, and ease of evolution [8]. There are two categories of concerns listed below:

- **Core concerns:** Primary concerns are the concerns which captures the centrally functionality of a module.
- **Crosscutting concerns:** The concerns that are capture system-level, peripheral requirements and cross multiple modules.

A classic enterprise application essential to report the crosscutting concerns that are Authentication, logging, resource pooling, administration, performance and storage management, data persistence, security, error checking and policy enforcement. All these concerns are crosscut many subsystems.

In OOPs technology, classes are used to implement primary concerns and these are related to core functionality and stated in problem domain. As in AOP, by contradicts the fundamentals of OOPs crosscutting concerns like security, synchronization, exception handling and scheduling all these crosscutting concerns are scattered or woven with the other code. However, OOP does not deal with practical or efficient concerns easily. AOP is a modular programming technology for separating concerns and for modularizing crosscutting concerns in well-identified software entities called aspects. AOP not replace OOP. AOP complements OOP by modularizing crosscutting concerns.

In OOP, the crosscutting concerns are coded in such a way that the code inside the method intermixed with the core logic of the method. This is called as code tangling. The logic code and the crosscutting concerns are tangled together i.e. spaghetti or tangled code. When the crosscutting concern code is used in multiple methods and multiple classes by using copy and paste resulted to the code that gets scattered throughout the application this approach is called code scattering [9].

This type of code scattering slows the maintenance of code, makes the errors and create problems for object-oriented applications to develop. Therefore, these terms crosscutting concerns and code scattering are the two main limitations of OOP. To conquer these limitations AOP defined the following terminologies.

III. TERMINOLOGIES INTRODUCED BY AOP

This segment presents the basic idea of AOP. Each novel programming come with the complete set of concepts and definitions.

A. Aspect

An aspect is a just like class in OOP. It is a modular unit of AOP and considered to implement a concern. Aspect consists of code, advice and different commands. Aspect languages built hierarchically and offer distinct mechanisms to define an aspect and organizing with a new system.

B. Join points

This a point in the program where concerns are crosscut the application i.e. an explicit point of the program. Typically, in the program there are various join points for single concern.

C. Advice

It is just a behavior, which is implemented by the join point. It means what action is going to be performed by the join point is mention by the advice.

D. Pointcut

A Pointcut is a set of join points. It deals with quantification mechanism, which is perform the action at many places in a program with a single statement.

E. Weaving

This is the technique of AOP in which core function modules combining with aspects. Many AOP languages define with either static weaving or dynamic weaving. In static weaving advice and base code compiling together while in dynamic weaving aspects are inserted at code loading and execute aspects with altering the interpreters.

IV. PROPERTIES OF AOP

AOP covers two main principles these are the Modularity and Separation of concerns and it includes two new properties that is Quantification and Obliviousness. In AOP, the modular unit defines the modularity for crosscutting concerns is the aspect. Moreover, the main characteristics of AOP projected by its properties and describe that whether the programming supports aspect oriented concept or not.

A. Obliviousness

Obliviousness is the influence of being oblivious. In AOP, the property obliviousness is the concept about the components, which are, not improve by aspects directly. When the obliviousness property is considered, it marks with following characteristics:

- Components are not conscious of the aspects that will ultimately crosscut them.
- No additional efforts done by the programmers for implementing the component's functionality to work AOP successfully with their benefits [10].

B. Quantification

Quantification is all about the skill to write separated and unique statements, which enrich to various non local places in a program. It states that "in programs P, whenever condition C arises, perform action A". If the quantification property considered, it marks with following characteristics:

- Aspects may crosscut an arbitrary number of components simultaneously.

For e.g. AspectJ, it supports dynamic quantification property.

To design an application in AOP both the properties are necessary. Obliviousness is considered the feature that makes AOP special [11]. On the other hand, quantification is also an essential property of AOP. The use of aspects to modularize crosscutting concerns supported by the obliviousness property enhances better separation of concerns in software development and simplifies the analysis, design and implementation of components [10].

V. BENEFITS OF AOP

AOP provides the system design and implementation in a new way. OOP promotes reusability and flexibility of code already.

So, what is the need of AOP? AOP is a programming paradigm that has all the benefits of OOP as well. Added to this, loose coupling and enable the application to use aspects without any change in application code.

In using AOP, focus is the business logic of application though at the same time weave the aspects to the business logic. One of the major benefits of using AOP is that just need to write the aspects once and then reuse it wherever it need in application. Therefore, AOP is very useful to reduce the complexity of the source code and make the code clean. The benefits of AOP involve:

A. Clean and clear Code—The most important advantage Of AOP is clean and clear code which is easy to maintain, read and with less errors. Making code easier to read is most significant because it is very comfortable for the new users and provide the speed for work. AOP permits moving the tangled code into the classes with clearer code [9].

B. Advanced modularization—AOP offers a mechanism to separately address a concern with minimum coupling. It results into an advanced modularized application in the presence of crosscutting concerns with the less amount of duplicate code [8].

C. System evolution at ease—AOP modularizes the each aspects and creates core modules oblivious to the aspects. Without affecting to the core modules, a new functionality is easily added easily by adding new aspect in the class. This results in faster response to new requirements [8].

D. Late binding of design decisions—By, using AOP it is possible to delay the design decisions and implement it later by just adding the aspects separately. Therefore, that developer can now focus on the basic core requirements [8].

E. Code reuse— With the loosely coupled implementation Each aspect considered as separate module and its key benefit is Code Reuse. All the core modules are not aware of each other only those aspects are aware of each other those have weaving rule [8].

F. Cheap costs of feature implementation—To implement a crosscutting concern, AOP makes it cheaper without affecting or modifying the original cost of the many modules. AOP allows the user to focus more on the concern of the module and the cost of the core need of the module is reduced [8].

VI. RESULTS AND DISCUSSION

This paper covers the various facets of Aspect Oriented Programming. With respect to the research analysis, these facets leads to some beliefs about AOP and resulted into that whether these beliefs true or false.

Table- I: Beliefs about AOP

Sr. No	*Beliefs*	*True*	*False*
1.	Rigid Program Flow	True	-
2.	Resolve Concern Problems	True	-
3.	Supports Messy Design	-	False
4.	Improves Clarity	True	-
5.	Breaks Encapsulation	True (In systematic and controlled way	-
6.	Replace OOP	-	False

As per the results mentioned in Table I, the conclusion of the research stated in next section.

VII. CONCLUSION AND FUTURE WORK

AOP is not complexed as it sound. AOP is an exciting and powerful paradigm that is great to use and it complements OOP. AOP is not replace the terms classes and objects in fact the application of it is organized with a set of classes. AOP conquers the problems occurred by code tangling and code scattering so it improves the productivity, reusability and the evaluation of the code. In AOP, every concern is addressed separately with minimal coupling, which, results a system with less duplicate code, modularised execution with the occurrence of crosscutting concerns. Aspects expand these classes by implementing crosscutting concerns to do job more effectively. In intelligent terms, this aids to the solo concern standard and use the open/shut standard efficiently with no repetition. In real-world terms, it will allow to spend more time adding value and less time doing tedious work. AOP is going to get a better frame of mind—and on time. In future, the research will deliver the realistic technique of the crosscutting concerns, aspects and weaving process.

ACKNOWLEDGMENT

We thank the Babasaheb Ambedkar Research and Training Institute (BARTI), Pune for funding our research.

REFERENCES

1. Tzilla Elrad, Robert E. Filman, and Atef Bader. 2001. Aspect-oriented programming: Introduction. Commun. ACM 44, 10 (October 2001), 29–32. DOI:https://doi.org/10.1145/383845.383853.
2. Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Videira Lopes, Jean-Marc Loingtier, John Irwin, Aspect-Oriented Programming, Published in proceedings of the European Conference on Object-Oriented Programming (ECOOP), Finland. Springer-Verlag LNCS 1241. June 1997.
3. Nicoara, Angela & Alonso, Gustavo. (2005). Dynamic AOP with PROSE. 125-138. tion Journals style)," IEEE Transl. J. Magn.Jpn., vol. 2, Aug. 1987, pp. 740–741 [Dig. 9th Annu. Conf. Magnetics Japan, 1982, p. 301].
4. Edsger W. Dijkstra. The structure of THE multiprogramming system. Communications of the ACM, 11(5):341–346, May 1968.
5. David Parnas. On the criteria for decomposing systems into modules. Communications of the ACM, 15(12):1053–1058, December 1972.
6. Ossher, H., Tarr, P., Using Multidimensional Separation of Concerns to (Re) Shape evolving Software ,Communication of the ACM, October 2001/Vol. 44, No. 10, pp 43-50.
7. Angela Hantelmann, Cui Zhang: "Adding Aspect-Oriented Programming Features to C#.NET by using Multidimensional Seperation of Concerns (MDSOC) Approach", in Journal of Object Technology, vol. 5 no. 4 Mai-June 06, pp. 59-83.
8. Ramnivas Laddad. 2009. AspectJ in Action: Enterprise AOP with Spring Applications (2nd ed.). Manning Publications Co., Greenwich, CT, USA. Bergmans, L. 1994. Composing concurrent objects. Ph.D. thesis, University of Twente.
9. Matthew D. Groves, AOP in .NET Practical Aspect-Oriented Programming Matthew D. Groves, Foreword by Phil Haack June 2013 , ISBN 9781617291142.
10. Filman, Robert. (2003). What Is Aspect-Oriented Programming, Revisited.
11. David Walker, Steve Zdancewic, and Jay Ligatti. 2003. A theory of aspects. In Proceedings of the eighth ACM SIGPLAN international conference on Functional programming (ICFP '03). ACM, New York, NY, USA, 127-139. DOI=http://dx.doi.org/10.1145/944705.944718.

AUTHORS PROFILE

Priyanka Sarode awarded B.Sc. degree in 2006 and MCA degree in 2009 from RTM Nagpur University, Nagpur. Currently she is pursuing Ph.D. in Computer Science and a research fellow at Inter Institutional Computer Center, Rashtrasant Tukdoji Maharaj Nagpur University, and Nagpur. Her research work is acknowledged by BARTI, Pune. Her main research work focuses on Programming languages, Aspect Oriented Programming. Mail Id: priya.s1011@gmail.com Mobile No. 9503869986

Ravikant Jugele is a M.Sc. in Computer Science from Marthwada University, Aurangabad in 1993. He also completed Ph.D. in Computer Science from Rashtrasant Tukdoji Maharaj Nagpur University. He is currently working as an Associate Professor in Department of Computer Science, Shivaji Science College, Congress nagar, Nagpur. Since 1995, his research interests include Multimedia and Hypermedia, cloud computing, Programming languages, Artificial Intelligence, Deep Technology and so on. Mail Id: rn_jugele@yahoo.com.