Metrics Analysis in Aspect Oriented and Object-Oriented Programming based on Exception Handling

[1] Shrikant Patel, [2] Sanjay Kumar, [3] Sandhya Katiyar, [4] Raju Shanmugam, [5] Shivangi Upadhyay
[1] Research Scholar, [2] Professor, [3] Associate Professor, [4] Professor, [5] Post Graduate Student
[1, 2, 4] School of Computing Science and Engineering, Galgotias University, Greater Noida, U.P., India
[5] Bhai Parmanand Institute of Business Studies, Delhi, India
[3] Department of IT, Galgotais College of Engineering & Technology, Greater Noida, U.P., India
[1]patelshrikant@rediffmail.com, [2]drkatiyarsanjay@gmail.com, [3]drkatiyarsandhya@gmail.com, [4]srajuhere@gmail.com, [5]shivangiupadhyay51@gmail.com

Abstract

Object Oriented Programming (OOP) is a programming paradigm that is focused on the objects and their behavior's concepts. It has been used for many years by software engineers' group. Design Patterns are the finest practice of OOP that has been gathered. OOP focuses on solving the problems in terms of real-world elements. But at some point, OOP has limitations and implementation which can be enhanced. AOP stands for Aspect Oriented Programming which is a programming paradigm that can overcome the limitation of the OOP and can achieve the successful results positively, but it can also cause the side effects in the code even with the successful results. Exception Handling is the mechanism which manages the run time errors to preserve the traditional flow of the application. However, the implementation of like these mechanisms with aspect-oriented programming (AOP) might lead to error containing scenarios. Using the AOP concept like joint cut, cross cut, aspect and many more we can prevent such scenarios. This paper tries to implement the code with exception handling with AOP concepts and identifies the advantages and disadvantages in collation with the traditional OOP implementation. This report presents the comparison of AOP with OOP through metrics analysis and provides the described information about this area of study

Keywords: *Object Oriented Programming (OOP), Aspect Oriented Programming (AOP), Exception Handling, Joint Cut, Cross Cut, Point Cut, Aspect, Weaving, Metric Analysis.*

1 Introduction:

Object oriented programming (OOP) is a programming worldview which holds the product business with its idea for such a long time. OOP had gotten well known in the dynamic programming dialects over the most recent couple of years. Encapsulation, Inheritance, Abstraction and Polymorphism are the four and primary spine of the OOP named as the standards of OOP. OOP was created to expand the reusability of the code and permits programming improvement to make adaptable applications. It gives numerous approaches to actualize the product productively and keep the standards of OOP, for example, low coupling, high attachment and some more.

After many research studies, it shows that some of the patterns can become better using AOP. It is a new programming paradigm that was introduced in late 1990's. Aspect Oriented Programming (AOP) was not introduced to overcome OOP but as an extension of OOP so that it can complement OOP. This paper is about the analysis of how exception handling code executes in Object Oriented Programming versus in Aspect Oriented Programming using metrics analysis.

Special case is an unforeseen occasion which intrudes on the typical progression of the program. These exemptions can be taken care of, so the program won't break. It is an article tossed at the runtime. Special case dealing with is an instrument that handles the runtime blunders, so the progression of program won't break when exemption happens. The combination of exception handling using OOP and AOP and their analysis is the main inspiration of this paper.

2 Object Oriented Programming

Object Oriented Programming is a programming paradigm which is developed by Alan Kay in the late 1960s. When the software application is developing using object-oriented analysis techniques based on the requirements provided by user is called the software development process. The software development model which focuses on objects and classes is called object-oriented programming model.

The basic unit of the OOP is an object which is also known as class instance. OOP has so many features through which they can perform loads of operations, can manipulate the data and more. Inheritance, Encapsulation, Abstraction and Polymorphism are few and main features of OOP. Good software which is developed using the OOPs concept must use them to achieve high cohesion, low coupling, code re-use in the software. OOP provides great features to make a good software application, but it has some limitations too.

Limitations of OOP

- No cross-cutting concerns: Object oriented programs needs to use as it is structure of code for all the modules which increases the development time as well as space complexity.

- Size: Object Oriented Program takes more space in terms of line of code and storage entire proceedings, and as a dependent document. Please do not revise any of the current designations.

The rest of the paper is written in following way. In Section-III we describe the aspect oriented programming. Section-IV gives the difference between aspect oriented programming vs. object oriented programming. Exception handling has been narrated in Section-V. Research methods have been proposed in Section-VI. Implementation of AOPS and OOPS programs have been given in Section-VII. Last but not the list, Conclusion and future research work have been proposed in Section-VIII.

3 Aspect Oriented Programming

Aspect Oriented Programming (AOP) supplements to Object Oriented Programming (OOP) by providing a different approach or unique way to think about the program structure. Aspect Oriented Programming is introduced around the late 1990s by [8] GregorKiczales and colleagues at Xerox PARC. They developed the explicit concept of AOP and further followed this with AspectJ which is an AOP extension of java. The major unit of modularity in AOP is aspect. There are numerous concerns which need to be managed along with the main business logic. These concerns are called cross-cutting concerns. There are few concerns like Logging, transaction handling, performance monitoring, security, etc. [24]. It involves join point, pointcut and advice.

A *Join point* is a specific place during the program execution such as execution of method; handle an exception and many more.

An *Advice* is considered as an action taken for the join point by an aspect. There are different types of advice: before, after and around.

Before Advice- Triggered before the join point method execution.

After Advice- Triggered after the join point method completely executed.

Around Advice- It is the most powerful, complex and important advice as it gives an option whether to execute join point or not.

A Point-cut is an expression who found matched with join point to decide whether an advice needs to be executed or not.

AOP provides a new perspective to manage cross-cutting concerns [10]. With the help of an aspect we can avoid scattering and tangling effects in the code. AOP is a vast topic in which researchers are working for better improvement of it as a technology.

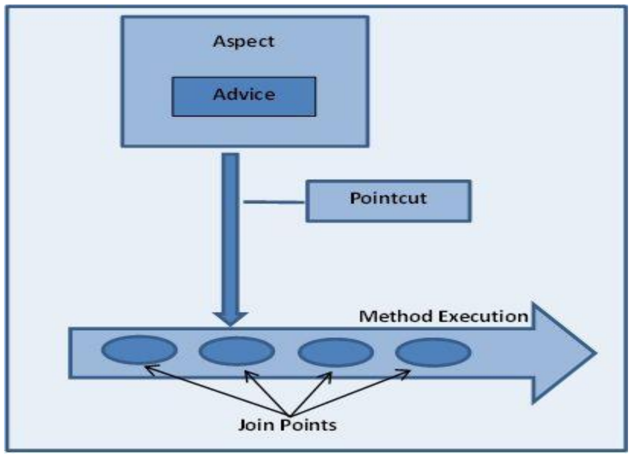


Fig. 1 Aspect Oriented Programming

Benefits of AOP

- Modularity: AOP helps in separating the core concerns from the non-core concerns by implementing the core concerns as aspects which reduces the scattering and tangling of code such as exception handling, etc.

- Performance: It improves performance as the functions are more concise.

- Maintenance: AOP provides an encapsulation for the re-usable code as an aspect which can easily be modified or removed without affecting the system which makes the maintenance easy.

- Functionality: Every aspect is independent and loosely coupled with another aspect in the application. This helps us to add or delete any aspect anytime without creating any issue in a program and without affecting the design of the application.

- Solution of cross-cutting procedure problem: We already introduced above in the limitation of OOP that OOP don't able to solve the cross-cutting procedure. But AOP comes up with the solution as we can create Aspect and can use one aspect for many modules.

A separate aspect is created for the logging procedure [7] so that whenever a logging procedure is called AOP will not create an object each time like in OOP.

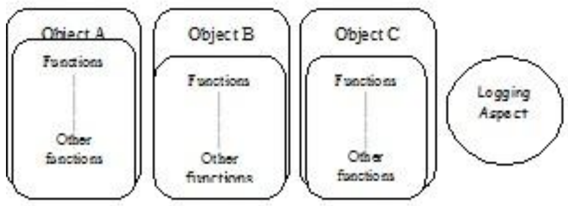


Fig. 2 Logging Aspect

Even though AOP has so many benefits it has some limitations too like a coin. There are some limitations mentioned below.

Limitations of AOP

- AOP community members are very few in numbers approximately between 2000-3000 programmers and only 14%-16% approx. of them are expertise to use AOP in OOP environment [15].

- The task of cross-cutting concern becomes very difficult to achieve when the complexity of program is high [15].

4 Aspect Object Oriented Programming vs. Oriented Programming

First, AOP and OOP is not competitor as AOP is originated from OOP paradigm. AOP is an extension of OOP which solves few problems OOP is not able to like cross-cutting concern.

The major difference between OOP and AOP is that Object Oriented Program focuses on to disintegrate the task into objects whereas Aspect Oriented Program focuses on to disintegrate the task into aspects.

In OOP, there is no logic of separating non-core concerns from core concerns which leads to code tangling [13] and code scattering.

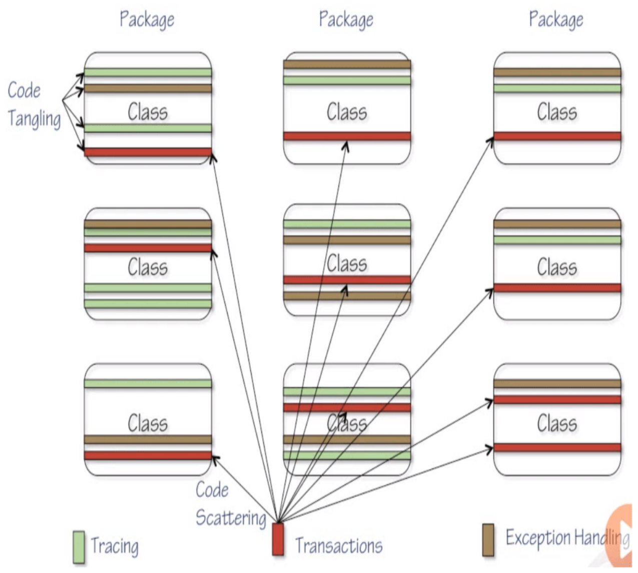


Fig. 3 OOP Code Distribution

In AOP, we use aspects which make the code neat, cleaner and loosely coupled. The main difference can be seen in the code itself that shows an implementation of some design pattern in OOP as well as in AOP.

We also list the table of metrics which will support the conclusion. We are using AOP with spring framework to produce a new research as how different results can a spring framework with OOP produce in comparison of spring framework with AOP.

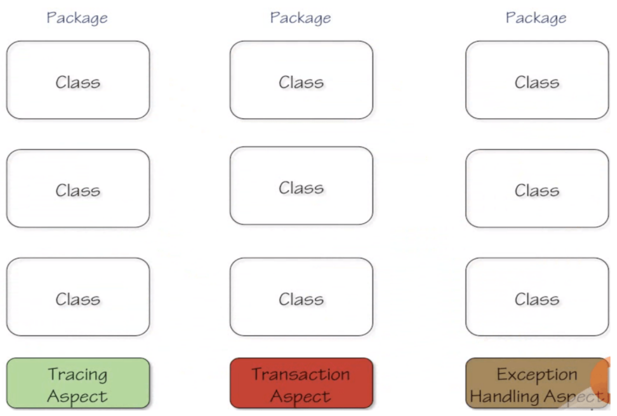


Fig. 4 AOP Code Distribution

5 Exception Handling

Exception is an abnormal event which interrupts or halts the traditional flow of the program. Whenever an exception is raised then exception object is created each time. This exception object contains load of debugging info about the method hierarchy, line on which the exception occurs, type of exception that occurred, etc. To handle the occurrence of exception we tend to do Exception Handling by the following methods:

- Throw – It is used when we want to generate an exception explicitly in our code. So, we throw a new exception.

- Throws – When the developer doesn't like to handle the exception then we use *throws* keyword, so the caller function or class knows beforehand that exception is going to be thrown.

- Try Catch Method – In this method we place the risky code in try block which is followed by catch block where exception is caught and handled.

We are analyzing how exception handling in OOP with spring is going to be different from exception handling in AOP with spring. In OOP with spring we are using the annotations like @Controller, @ExceptionHandler, etc. whereas in AOP with spring we are using the annotations like @AfterThrowing, @BeforeThrowing, etc. for exception handling.

6 Research Methods

Implementation of the patterns with AOP can be considered a decent viewpoint to solve the pattern problems. But it can also cause some side effects that should be interesting thing to analyze.

We are going to analyze how the scenario changes when we approach with AOP versus approach with OOP. It will also highlight the benefits and drawbacks of using it for the patterns.

Metrics

Metrics are necessary to use to evaluate the patterns implementation with AOP and OOP. Metrics let the engineer to examine the properties in terms of GRASP principles. Those metrics are explained below:

- Coupling: It is known to measure the complexity of interactions between one modules of the program to the other modules. It analyzes the program and determines the dependencies of a module has with other module and tell whether an application is loosely coupled or tightly coupled.

- Cohesion: It is known to measure the interactivity in between the module. It determines the responsibility of a module is correctly assigned or the behavior of a module is fine. A module with high cohesion can't function big amounts of tasks.

- Size: It evaluates how large the classes/module is in terms of LOC and token count.

▪ Lines of code (LOC) – It includes each line of the program containing header declaration, etc. except comment and blank line.

▪ Token Count – LOC has few issues as it is inconsistent because the difficulty of coding level of some lines is high than the other lines. A program can be contemplated as a series of tokens which is to be classified as either operator or operand [7]. Operators are symbols or keywords that are found in a program and tell the compiler to perform the specific operations. +, -, /, *, %, ++, -- are the signs collectively known as Arithmetic Operators , ==, <, >, =, <=,>= are the signs collectively known as Relational Operators, !, &&, || are the signs collectively known as Logical Operators , Bitwise Operator &, |, ^ are the signs collectively known as Bitwise Operators, =, +=, -=, %=, /= are the signs collectively known as Assignment Operators and size of, cast, etc. are collectively known as Misc Operators. These are the types of operator a program can consist. Symbols consists of braces like {}, [] & (). Operands can be considered as the variable which stores the value. So, we calculate the size of vocabulary, program length, volume, estimated program length, effort and many more.

Spring with AOP

Spring AOP is applied in pure java. There is no requirement of special compilation process for this. The Class Loader Hierarchy does not need to be controlled by Spring AOP and thus it is suitable for use. Now a days AspectJ library in spring framework are influencing the software market. We are doing our analysis of Aspect Oriented Programming with the spring framework. Spring uses proxy-based mechanism. It will make a proxy object which enfold the target object and will use an advice which relevant to function call. A proxy-object can be created by two different approaches:

1) Manually through proxy factory bean,

2) Through auto proxy configuration in the xml file. They will get destroyed after the execution takes place.

It uses XML based configuration for applying AOP. It uses annotations which are interpreted by a library provided by AspectJ. Declarative transaction management is the important service. It permits the users to implement custom aspects, go with the use of OOP with AOP. The user just writes the dependencies of AspectJ then the user will able to use the annotations of Aspect like @Aspect.

7 Oops &Aops Program Implementation

Implement without Aspect

https://github.com/Shivangi-1007/Exception-Handling-without-AOP

Implement with Aspect

https://github.com/Shivangi-1007/Exception-Handling-with-AOP

Comparison between OOP and AOP program using Halstead Software Science Metrics Method

Here we are going to mention some formulae that we used to do the comparison between AOP and OOP.

- Size of the vocabulary:

$\eta = \eta_1 + \eta_2$

Where η = Program vocabulary.

η_1 = number of unique operators.

η_2 = number of unique operands.

- Length of the program

$N = N_1 + N_2$

Where N = Program Length

N_1 = Total no. of repetition of operators.

N_2 = Total no. of repetition of operands.

- Volume

$V = N * \log_2 \eta$

- Program Level

$L = V^* / V$

Where V = Volume of the program

V^* = Potential Volume

$0 <= L >= 1$

Here, L=1, Since program is written at highest possible level.

- Difficulty

$D = 1 / L$

- Effort

$E = V/L = D*V$

- Estimated Program Length

$\check{N} = \eta_1 \log_2 \eta_1 + \eta_2 \log_2 \eta_2$

- Required Time

$T = E/S$

Where T = Time required for a program's effort

S = Stroud Number (It is set to 18 for software scientists)

E = Effort

- Estimated Program Level

$\acute{L} = 2\ \eta_2 / (\eta_1, N_2)$

Where $\acute{L}$ = Estimated program level

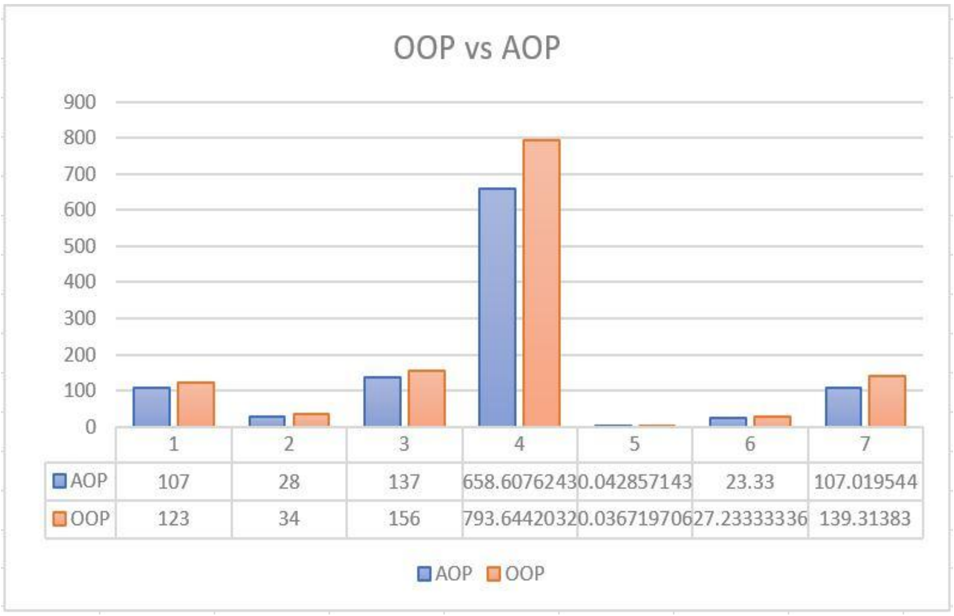
Table 1: Metrics Details (OOP)

METRICS	**OOP**
LOC	123
Ŋ	34
N	156
V	793.644203235
L	0.0367197062
D	27.233333364742
E	21613.57715969416
Ň	139.31383
T	1200.75

Table 2: Metrics Details (AOP)

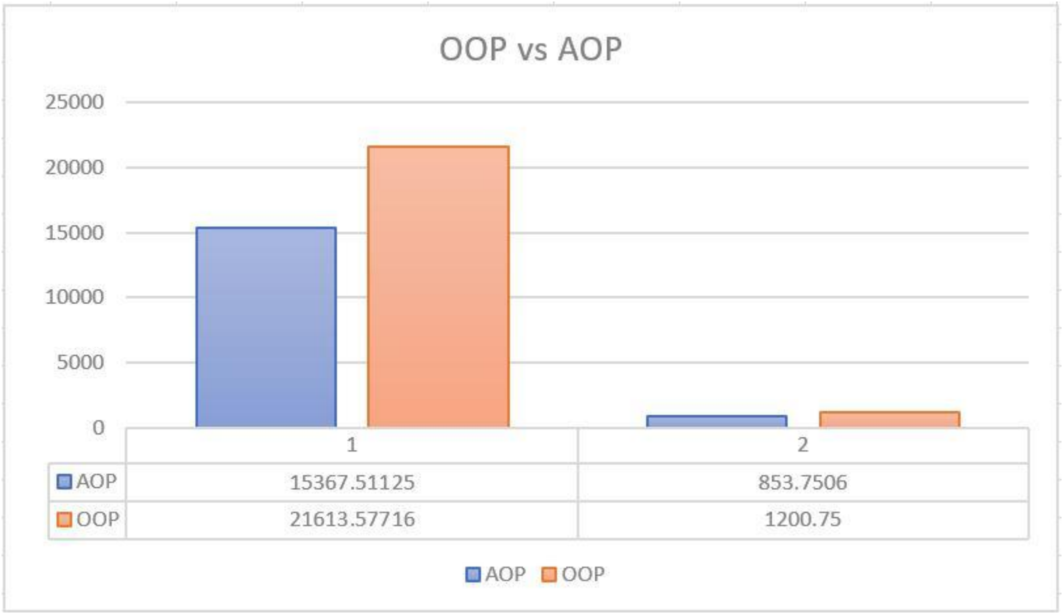
METRICS	**AOP**
LOC	107
Ŋ	28
N	137
V	658.607624322
L	0.0428571428
D	23.33
E	15367.51125467002
Ň	107.019544
T	853.7506

In the Table 1, we observe the metrics details about the program written in OOP programming language. In the Table 2 we observe the different metrics details about the program written in AOP programming language.



	1	2	3	4	5	6	7
🟦 AOP	107	28	137	658.6076243	0.042857143	23.33	107.019544
🟧 OOP	123	34	156	793.6442032	0.036719706	27.23333336	139.31383

Fig. 5 Comparison between OOP and AOP



	1	2
AOP	15367.51125	853.7506
OOP	21613.57716	1200.75

Fig.6 Comparison between OOP and AOP

Fig. 5 and Fig. 6 show the comparison between OOP and AOP based on metrics details. As we can see clearly that AOP is showing more effective and good result than OOP. We did this comparison for the small and simple program which will not show the major difference between OOP and AOP but when the program is large and more complex than the difference between OOP and AOP will be huge.

8 Conclusion & Future Research Work

In this paper we have analyzed the metrics of Aspect oriented program and Object-Oriented program. We observed that AOP is showing better performance than OOP even with the simple code of program. We conclude that AOP is better programming paradigm than OOPs as AOP can overcome the issue of cross-cutting concern.

OOP has been present for the so many decades while AOP is just started to be known in the market as a dominating language. AOP is not competing with OOP as AOP is originated from OOP. AOP is a resolving to those issues that OOP couldn't solve. The major difference between AOP and OOP is that OOP concentrates onto the basic unit which will encapsulate the data and behavior while AOP concentrates on cross-cutting concerns as separating the core concern from non-core concern. This makes the program free from the effect of code tangling and code scattering. We just compared AOP with OOP on basis of exception handling. But in future, AOP might expand its market in some areas of technology and more in research field. There are still many new things to discover about AOP yet. AOP has so much prospective which hasn't discovered yet which makes the researchers more to analyze this programming language.

References

1. George William Howarth, "An Investigation into Aspect-Oriented Programming," School of Computer Science. The University of Manchester, MSc Dissertation 2012.

2. Gulia P., Khari M. , Patel S., “Metrics Analysis in Object Oriented and Aspect Oriented Programming”, Recent Pattern on Engineering , 2019.
3. Pavol Baca and Valentino Vranic, "Replacing Object-Oriented Design Patterns withIntrinsic Aspect-Oriented Design Patterns," Faculty of Informatics and Information Technologies. Slovak University of Technology, Bratislava, 2011.
4. Elisa Y Nakagawa, Fabiano C Ferrari, Mariela M.F Sasaki, and José C Maldonado, "Anaspect-oriented reference architecture for Software Engineering Environments," The Journal of Systems and Software, vol. 84, 2011.
5. Gulia P., Khari M. , Patel S., “Comparative Analysis of Object Oriented programming and Aspect Oriented Programming approach ”, 2015 International Conference on Computing for Sustainable Global Development, INDIACom 2015 (2015).
6. Peng Wang and Xiaochun Zhao, "The Research of Automated Select Test Cases for Aspect oriented Software," Information Engineering Research Institute, 2012.
7. Walter Binder, DaniloAnsaloni, Alex Villazón, and Philippe Moret, "Flexible and efficientprofiling with aspect-oriented programming," Concurrency and Computation: Practice AndExperience. Wiley Online Library, 2011.
8. Ryan M Golbeck, Peter Selby, and Gregor Kiczales, "Late Binding of AspectJ Advice,"Lecture Notes in Computer Science LNCS, vol. 6141, 2010.
9. Preeti Gulia, Manju Khari and Shrikant Patel, “Metric Analysis in Object Oriented and Aspect Oriented Programming.” Bentham Science Journal Vol. 13, Issue 2, 2019.
10. G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C.Videira Lopes, J.-M. Loingtier, J. Irwin, “Aspect Oriented Programming”, In Proc. Europ. Conf. on Object-Oriented Prog. (ECOOP), Finnland, Springer Verlag LNCS 1241, June 1997.
11. I. Boticki, M. Katic, S. Martin, “Exploring the Educational Benefits of Introducing Aspect-Oriented Programming into a Programming Course,”, IEEE Transactions on Education, vol.56, no.2, pp.217-226, May 2013.
12. T. Zukai, P. Zhiyong, “Survey of Aspect-Oriented Programming Language, Journal of Frontiers of Computer Science and Technology, 2010, vol.4, no.1, pp 1-19.
13. D. Zhengyan, “Aspect Oriented Programming Technology and the Strategy of Its Implementation,” In Proceedings of International Conference on Intelligence Science and Information Engineering (ISIE), 2011, pp.457,460, 20-21.
14. R. D. Dechow (2005), “Advanced Separation of Concerns and the Compatibility of Aspect-Orientation. accessed [26/8/2013] [online].
15. Cristina Lopes, Jim Hugunin, Mik Kersten,Martin Lippert, Erik Hilsdale and Gregor Kiczales “Using AspectJ^{TM} For Programming- The Detection and Handling of Exceptions”. ICSE '00: Proceedings of the 22nd international conference on Software engineering, June 2000.
16. Khalid Aljasser, “Implementing design patterns as parametric aspects using ParaAJ: the case of the singleton, observer and decorator design patterns.” Computer Languages, Systems & Structures 45 (2016): 1-15.
17. Heba A. Kurdi “Review on Aspect Oriented Programming.” International Journal of Advanced Computer Science and Applications, Vol. 4 No. 9, 2013.
18. Christoph Seidl,, Schuster Sven and Schaefer Ina. “Generative software product line development using variability-aware design patterns.” ACM SIGPLAN Notices. Vol. 51, No. 3 ACM, 2015
19. Ergin, Huseyin, Eugene Syriani and Jeff Gray. “Design pattern-oriented development of model transformations.” Computer Languages, Systems & Structures 46(2016):106-139.
20. design and prototyping of service-oriented applications with design patterns.” Computer Languages, Systems & Structures 46(2016): 140-166.
21. Steimann, Friedrich. “Why most domain models are aspect free.” 5th Aspect-oriented modelling workshop AOM at UML 2004.

22. Steimann, Friedrich. “Domain models are aspect free.” International Conference on Model Driven Engineering Languages and Systems. Springer, Berlin, Heidelberg, 2005.
23. Rashid, Awais and Ana Moteira. “Domain models are not aspect free. “International Conference on Model Driven Engineering Languages and Systems, Springer, Berlin, Heidelberg, 2006.
24. K. Lieberherr, D. Orleans, J. Ovlinger, “Aspect-oriented programming with adaptive methods,” ACM Communication, 2001, vol. 44, no. 10, pp 39-41.
25. M. Ali, M. Babar, L. Chen, K. Stol, “A systematic review of comparative evidence of aspect-oriented programming,” Information and Software Technology, 2010, vol.52, no.9, pp. 871-887.
26. https://en.wikipedia.org/wiki/Aspect-oriented_programming#cite_note-1.