International Journal of Engineering Applied Sciences and Technology, 2016
Vol. 1, Issue 3, ISSN No. 2455-2143, Pages 50-54
Published Online December – January 2016 in IJEAST (http://www.ijeast.com)

ELUCUBRATION OF ASPECT ORIENTED PROGRAMMING

Shivam Verma, Ved Prakash Singh, Vaibhav Varshney
G.L. Bajaj Institute of Technology

Sahil Aggarwal
Asst. Professor
G.L. Bajaj Institute of Technology

ABSTRACT - Aspect oriented programming is an addition to OOP rather than a replacement of it. This programming approach is considered to be more efficient than just OOP when handling properties of non-functional component like, logging, synchronisation, tracking, security, data validation fault tolerance and exception handling. This programming pattern actually complements the primitive Object Oriented Programming Paradigm to enhance its efficiency while handling critical issues like cross cutting concerns. This research document throws some light on the use of Aspect oriented programming pattern for achieving better results in modularity of the code. It concludes that Aspect oriented software development is essentially an attempt to modularize those concerns that we can't modularize very well with traditional object oriented languages or statement oriented languages and it deserves more attention being a promising programing pattern. The paper has been divided into individual sections which particularly explain every corner of this programming approach.

I. INTRODUCTION

The term Aspect oriented programming was coined by et. Al Kiczales in 1997[1]. He had a fore sight of the problems that are yet to be faced by programmers of the modern era and proposed a solution for the same.

Nowadays we are building; larger, complex much more distributed systems and these very large systems have lots of concerns that cannot compete with each other like security, auditing, tracking, synchronization, exception handling, fault-tolerance, logging etc.

There have been found many programming problems for which neither procedural nor object-oriented programming techniques are sufficient to clearly capture some of the important design decisions that the program must implement. Aspect Oriented Programming is developed in order to get better results modularity and separation of concerns (SoC) when used in collaboration with Object Oriented Programming.
AOP when used along with the primitive Object oriented programming pattern has proved to deliver better results by making the program less superfluous, less scattered and tangled as compared to Object Oriented Programming Paradigm solely.

Even though this pattern sounds a promising programming methodology, there have been doubts about applicability and effectiveness of AOP. Therefore this paper presents an in-depth study regarding issues discussed further in this paper, and how AOP proves to be a potential solution to these problems.

II. IMPLEMENTATION OF AOP

Dissimilar to the programming approach of the primitive programming paradigms, AOP provides external support for enhancing the quality of the code. It provides explicit support for modularizing programs; rather than scattering the code related to a non-functional requirement or a concern throughout a program [2]**.**

These are some essential terminologies required for better understanding of this advanced programming pattern:

An aspect is modularised implementation for a crosscutting concern. It merges the scattered code that of a crosscutting concern in a module. A process in which an aspect is added to an object is called weaving. It can be executed in the compiling time or during the running time of the program [3].

A well-defined position in the program as throwing an exception or invoking a method is called a Join point [4]**.**Crosscutting Concerns are the aspects of a program that affect other concerns. These concerns often cannot be cleanly decomposed from the rest of the system in both the design and implementation, and can result in either scattering (code duplication), tangling (significant dependencies between systems), or both.

A class of methods/procedures that can alter other methods and the code whose execution is triggered when a join point is reached is called an ADVICE whereas Pointcut is a set of join points which executes the corresponding advices whenever reached.

2.1 AspectJ

AspectJ [5][6] is the most efficient and widely used tool that AOP developers use for software production. It is an extension to the already present and in use java programming language and uses syntax similar to that of java. It supplied as a part of java software development kit (SDK) from the official website. All the java programs are valid in AspectJ in addition to the special extended version of a class which is called an Aspect [7]. In addition to the components of a standard class an aspect has some additional entities such as pointcuts and advices. In order to generate the java byte codes AspectJ needs a special compiler. The byte code so generated has more difference as compared to the general java byte code files.

III. ADVANTAGES OF AOP

According to Kiczales [1], there are many programming problems for which neither procedural nor object- oriented programming techniques are much sufficient to clearly capture some of the important design decisions the program must implement.

AOP has presented itself as a promising approach and as an effective solution for conventional programming approaches problems.

According to R. Laddad [8], using AOP for implementing software systems will certainly enhance software quality in many ways including:

3.1 Improved modularity

AOP provides better modularization by combining the code that deals with the same aspect in one module avoiding the duplication of crosscutting concern. It also leads to a better software development process because each developer could use their own skills with the module.

3.2 Lesser line of code

By use of AOP we can use the same set of code for each time we require that module. This enhances the space complexity by providing improved reusability because it prevents intermixing of crosscutting concerns from the core concerns and by creating reusable aspects.

3.2.1 Non-intrusive conditional analysis

Unlike the conventional programming techniques this pattern does not waste time and space checking whether the functionality is needed by the object or not.

3.2.2 Concordant implementation

In contrast to the traditional implementation of crosscutting concerns, AOP provides concordant implementation by having each aspect handled once.[11]

3.2.3 Better skill transfer

This programming pattern inculcates features like reusability and transferability. Therefore, developers learning time and cost will be reduced even if they want to learn other languages at the same time as the core concerns and design pattern are common to all.

This technique allows the software developers to implement a wide range of contracts. While using mock objects, testing of software technology is used in the operation of this technique. Some of the conditions are not tested because of the complications which results into network failure. This programming pattern makes the make testing process more convenient without much need of alternation in the core code structure

IV. PROS AND CONS

Extensive research has been done regarding this approach which claims it to have great benefits in software engineering.

Ali et al. [9] explained the comparative facts of this paradigm. They have discussed the advantages and shortcomings of it in detail on the basis of the following standards:

Performance, code size, modularity, cognition, language mechanism, evolvability. Each of these standards is further explain in this section.

The use of this paradigm improve execution performance by decreasing the response time and minimising the usage of memory and hardware cost which in turn provided a better space and time complexity of the program software. Though the results were unnoticeable when tests were carried in UNIX based operating system.

Kiczales [1] stated that inculcating this approach by replacing the conventional technique of object oriented paradigm, would create a considerable reduction in the code size because of separation of crosscutting concern as discussed in the prior sections.

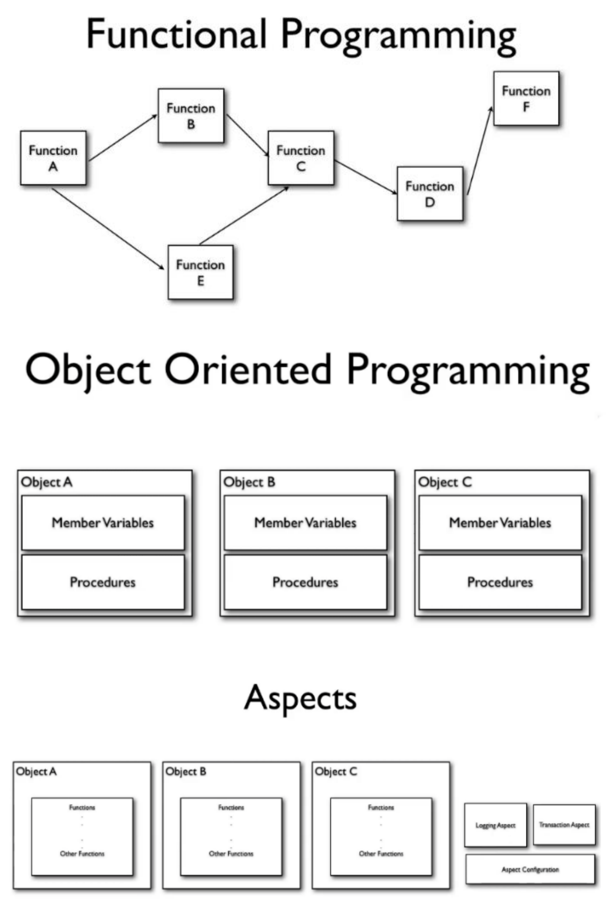

Fig B: The structural difference between AOP, OOP and FP.

An elaborated research [12] concluded that there was a significant reduction in the size of code approximately by 39.5% which implies that there was a reduction in the line of code as well, because of separation of crosscutting concern. This connotes that AOP is actually effective in reducing the size of code positively most of the time.

Due to the reason that this programming paradigm is not so popular yet, the programs written according to this approach were not adapted to because when looking through the development time and understandablity the results were insignificant.

Evolvablity: - This means the programs ability to allow changes to be done in the programs according to the requirement. This technique yielded a better result for this standard as compared to the use of conventional technique of object oriented paradigm.

The overall structure of the program is modernised by replacing the conventional technique of object oriented paradigm, because the way AOP deals with the code is different from the prevailing approach. The use of this technique has improved the modularity of the code significantly by separating the crosscutting concerns which is done by placing them in a separate aspect.

Each criteria was studied and was concluded with one of the four possible results:

- Positive: when they note improvement of the criteria with AOP compared to non AOP implementation.
- Negative: when involvement of introducing aspects are not profitable in the context.
- Mixed: when the effect were positive and negative in some cases.
- Insignificant: when the difference is not noteworthy.

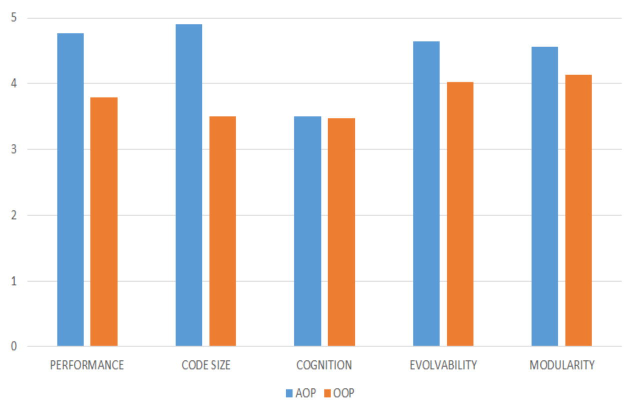

Fig C: Graphical Comparison of different Standards between AOP(left) and OOP(right)

V. CHALLENGES

AOP has not been adapted widely because of some disadvantages and challenges associated with it. Programmers are required to read the code and analyse that what exactly has to be done to prevent errors. The debugging process is much harder in AOP,[7] as compared to OOP because the programmers face difficulties in understanding the crosscutting concerns because programmers are required to perceive the core module implementation details.

Moreover, if there is a logical error formed in expressing crosscutting concerns, it results in the widespread failure of the program. Aspect-oriented programming is patented, [1] and thus is not freely implementable.

According to Luca and Depsi [10], AOP faces some challenges as a new programming technique. There are approximately 1900-2000 programmers who are aware of the AOP concepts and are in AOP community worldwide and only few of them are experienced enough to use this approach in OOP environment. AOP has provided some new aspects and a new approach in to programming. It provides better modularity (Separation of concerns) but when the system reaches to a certain extent of complexity, such separations are hard enough to obtain.

VI. CONCLUSION AND FUTURE SCOPE

As many studies have been conducted to look into the consequences of AOP compared to primitive technique on features regarding software development process and the software which have already been developed in the late 90's.

Here it is found that most of the reviewed studies have resulted into both positive and of no significant effect on this paradigm when compared to the traditional approaches. As thoroughly discussed in the previous sections, AOP provides better modularity plus it helps to reduce the code size by separating cross cutting concerns and placing them into a separate Aspect. This facilitates the user to modify the program according to the demand. The effect on the understandably and development time were not significant but the implementing this technique gave the user a considerable increase in the performance of the code software by improving the space and time complexity of the program

However, crosscutting concerns are difficult to understand because programmers are required to perceive the core module implementation details. Though this programming technique is not so in use yet but, deserves to take its righteous place in the programming community. Only then could researchers study AOP effectively and efficiently.

VII. REFERENCES

[1] 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.

[2] S.R. Chidamber, C.F. Kemerer, “A metrics suite for object oriented design,” IEEE Transaction on Software Engineering, 1994, vol. 20, no. 6, pp. 476–493

[3] 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

[4] Heba A. Kurdi “Review on Aspect Oriented Programming” Computer Science Department Imam Muhammad Ibn Saud Islamic University Riyadh, Saudi Arabia pp: 22-24

[5] G. Kiezales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, W.G. Griswold, “An overview of AspectJ,” In Proceedings of the 15th European Conference on Object-Oriented Programming (ECOOP’01), 2001, pp. 327–353.

[6] AspectJ: an aspect-oriented extension to Java. http://www.eclipse.org/aspectj/doc/released/progguide/language.ht ml

[7] T. Xie, J. Zhao, “A framework and tool supports for generating test inputs of AspectJ programs,” In Proceedings of AOSD, 2006, pp. 190– 201.

[8] R. Laddad, “Aspect-oriented programming will improve quality,” IEEE Software, 2003, vol.20, no. 6, pp. 90-91.

[9] 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.

[10] L. Luca, I. Despi, “Aspect Oriented Programming Challenges,” Anale Seria Informatica, 2005.vol. 2, no. 1, pp. 65-70.

[11] S.R. Chidamber, C.F. Kemerer, A metrics suite for object oriented design, IEEE

Transactions on Software Engineering 20 (1994) 476–493.

[12] L. Hatton, How accurately do engineers predict software maintenance tasks?, IEEE Computer 40 (2007) 64–69