

Review on Aspect Oriented Programming

Heba A. Kurdi

Computer Science Department
Imam Muhammad Ibn Saud Islamic University
Riyadh, Saudi Arabia

Abstract—Aspect-oriented programming (AOP) has been introduced as a potential programming approach for the specification of nonfunctional component properties, such as fault-tolerance, logging and exception handling. Such properties are referred to as crosscutting concerns and represent critical issues that conventional programming approaches could not modularize effectively leading to a complex code. This paper discusses AOP concept, the necessity that led to it, how it provides better results in code quality and software development efficiency, followed by stating challenges that developers and researchers face when dealing with this approach. It has been concluded that AOP is promising and deserves more attention from developers and researchers. However, more systematic evaluation studies should be conducted to better understand its implications.

Keywords—Aspect Oriented Programming; software engineering; AspectJ

I. INTRODUCTION

This A typical program code is composed of several distinct components. Each of these components is responsible for accomplishing a core function required by the system. Some concerns, though, such as error handling, security and synchronization, are important for the entire system and they therefore crosscut multiple components. Implementing these crosscutting concerns is considered to be a challenging issue that conventional programming approaches, such as Object-Oriented Programming (OOP) and Procedural-Oriented Programming (POP), can not modularize very effectively. Lack of code modularity usually results in a tangled and complex code. As a result, Aspect-Oriented Programming (AOP) has recently emerged as a promising new approach to handle this issue. The term was coined by Gregor Kiczales in 1997 [1] as a complement to the OOP rather than as a replacement to it [2].

From the linguistic meaning of the word “aspect”, a general idea of the technical meaning would arise. AOP is a programming approach that aims to solve crosscutting concerns throughout better modularization of the code. It enhances system features such as modularity, readability and simplicity by better handling of crosscutting concerns [3]. Based on this definition, it is clear that AOP makes a clear distinction between two types of concerns in the software development process:

- Primary concern: represents real world components or objects. In OOP, a class represents each of these components.
- Crosscutting concerns: refers to a programme design feature that is required by multiple software

components. Therefore, its implementation is scattered and/or repeated among them, severely affecting code modularity [4].

For instance, in a banking system, primary concerns include customer and account management, statement generation, transaction tracking ... etc. These concerns are usually implemented as procedures (operations), or classes in conventional programming approaches, i.e. OOP and POP. Examples of crosscutting concerns would include exception handling, authentication and security aspects, which are usually considered essential parts of many procedures or classes in conventional approaches. Therefore, they are handled in multiple locations within the same program, causing a drastic decrease in the quality, readability and modularity of the software [12]. Aspects are treated differently in AOP. They are considered an extended version of the class with additional features [5]. Figure 1 shows the central concepts in each of the three programming approaches and how they are related to each other.

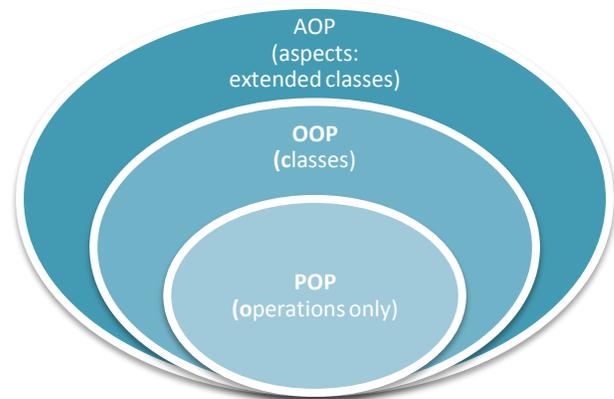


Fig. 1. The relationship between POP, OOP and AOP

Even though an increasing number of programmers and software engineers started adopting the AOP approach, a lot of concerns and challenges are still hindering wider adoption [2]. Therefore, this paper reviews the state-of-the-art in AOP and sheds some light on its related issues, starting with its terminologies and implementation approaches in section 2. The needs that led to the introduction of AOP and its potential benefits are presented in section 3. Section 4 then goes on to provide an overview of previous works that conducted evaluation studies of AOP. In section 5, possible threats and challenges of AOP are discussed and, finally, section 6 provides the conclusion, summarizing the paper and spotting some future research directions.

II. AOP IMPLEMENTATION APPROACHES

Unlike traditional programming approaches AOP provides explicit support for modularizing programs; rather than scattering the code related to a non-functional requirement or a concern throughout a program [19], developers can place it within a separate segment [15]. This required introducing new programming concepts and terminologies such as:

- **Crosscutting concern:** is a purpose that a program wants to achieve. However, this purpose should be scattered among many classes or methods.
- **Aspect:** is a modularized implementation for a crosscutting concern. It amalgamates the distributed code that of a crosscutting concern in one module.
- **Join point:** is a well-defined position in a program, such as throwing an exception or invoking a method.
- **Advice:** is a class of functions that can modify other functions. It is applied at a given join point of a program.
- **Pointcut:** is a general term for a set of joint points whenever reached the corresponding advices will be executed.
- **Weaving:** is the process in which an aspect is added into an object. It can be executed in the compiling time or during the running of the program [6].

There are two approaches for implementing AOP:

A programming language that has been developed specifically for AOP, such as AspectJ: AspectJ [22][23] is the first and most popular tool that AOP developers use for creating software. It is an extension for the Java programming language and uses a Java-like syntax [13]. It is available for download as part of Java software development kit (SDK) that supports it from the official website. All Java programmes are valid in AspectJ, in addition to a special extended version of a class, which is called an aspect [17]. An aspect contains all components of a regular class, as well as some additional entities such as pointcuts and advices [4]. AspectJ needs a special compiler to generate Java byte code. The java class file generated by AspectJ compiler has no difference compared to general Java byte code files [6]. Figure 2 presents an example of AOP in AspectJ.

- Techniques provided by already available programming languages to support aspect implementation:
- Many programming framework have released additions to support ASP[18][20], such as .NET [8] and Spring. Figure 3 illustrates an example of ASP in Spring AOP.

A detailed survey of AOP implementation techniques is provided in [6].

III. AOP ADVANTAGES

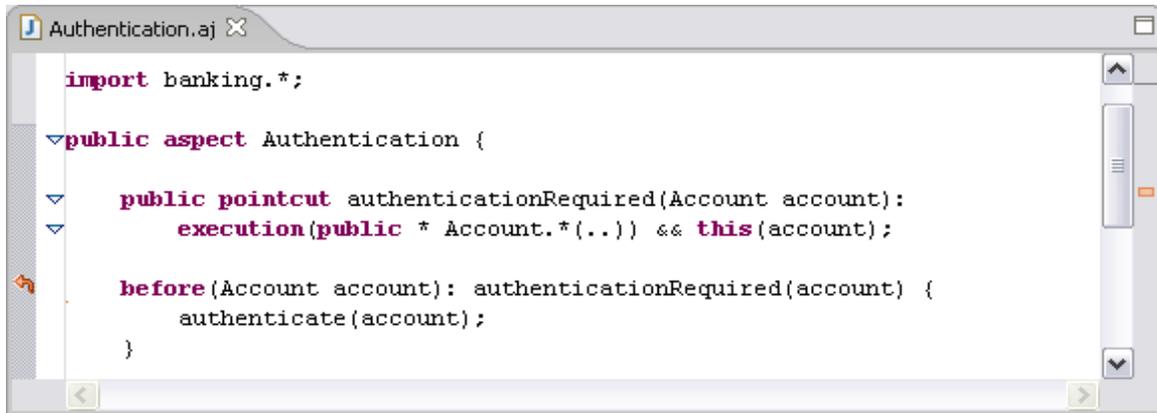
According to Kiczales [1] the OOP and POP have many programming problems that did not allow these approaches to

clearly capture some design elements which are important for software implementation. Therefore, AOP presented itself as a promising approach and as a solution for conventional programming approaches problems. However, solutions provided by AOP do not necessarily come in terms of lower compilation time or less memory usage. Rather, according to Laddad [9], using AOP for implementing software systems will certainly enhance software quality in many ways including:

- **Clear responsibilities for individual modules:** AOP offers better modularisation, by gathering the code that deals with the same aspect in one module avoiding the redundancy of crosscutting concerns. This also leads to a better programming development process because each developer could use his/her expertise with the module he/she knows better.
- **Consistent implementation:** Unlike traditional implementations of crosscutting concerns, which are conspicuous in their inconsistency, AOP provides consistent implementation by having each aspect handled once.
- **Improved reusability:** AOP isolates core concerns from the crosscutting ones, enabling more mixing and matching, and therefore improving the overall reusability in both modules. In contrast, traditional methods do not have this kind of separation between concerns.
- **Improved skill transfer:** The concepts of AOP are reusable and transferable. Therefore, developers training time and cost will be minimised even if they need to learn more than one language. This is because core concerns and design patterns are universal. However, this is not the situation in other frameworks, where developers have to learn from the beginning each time, wasting considerable time and money on training.
- **System-wide policy enforcement:** AOP allows programmers to enforce a variety of contracts and provide guidance in following “best” practices by creating reusable aspects.
- **Logging-fortified quality assurance:** The disability of replicating a bug is one of the major disappointments for traditional methods’ developers, because it is such a ponderous process and thus barely used. On the other hand, AOP enables quality-assurance persons to attach the bug paper with its log, easing the reproduction of the behaviour by the developer.
- **Better simulation of the real world through virtual mock objects:** Software quality testing is enhanced in AOP application by using mock objects. Some scenarios often are not tested because of their complexity that requires an effort to simulate faults such as a network failure. AOP makes the difficult and cumbersome testing process easier without the need to compromise the core design for testability.
- **Nonintrusive what-if analysis:** Dissimilar to non-AOP approaches, AOP does not waste time and space by

checking whether functionality is needed by running what-if analysis every time before changing the system

behaviour.

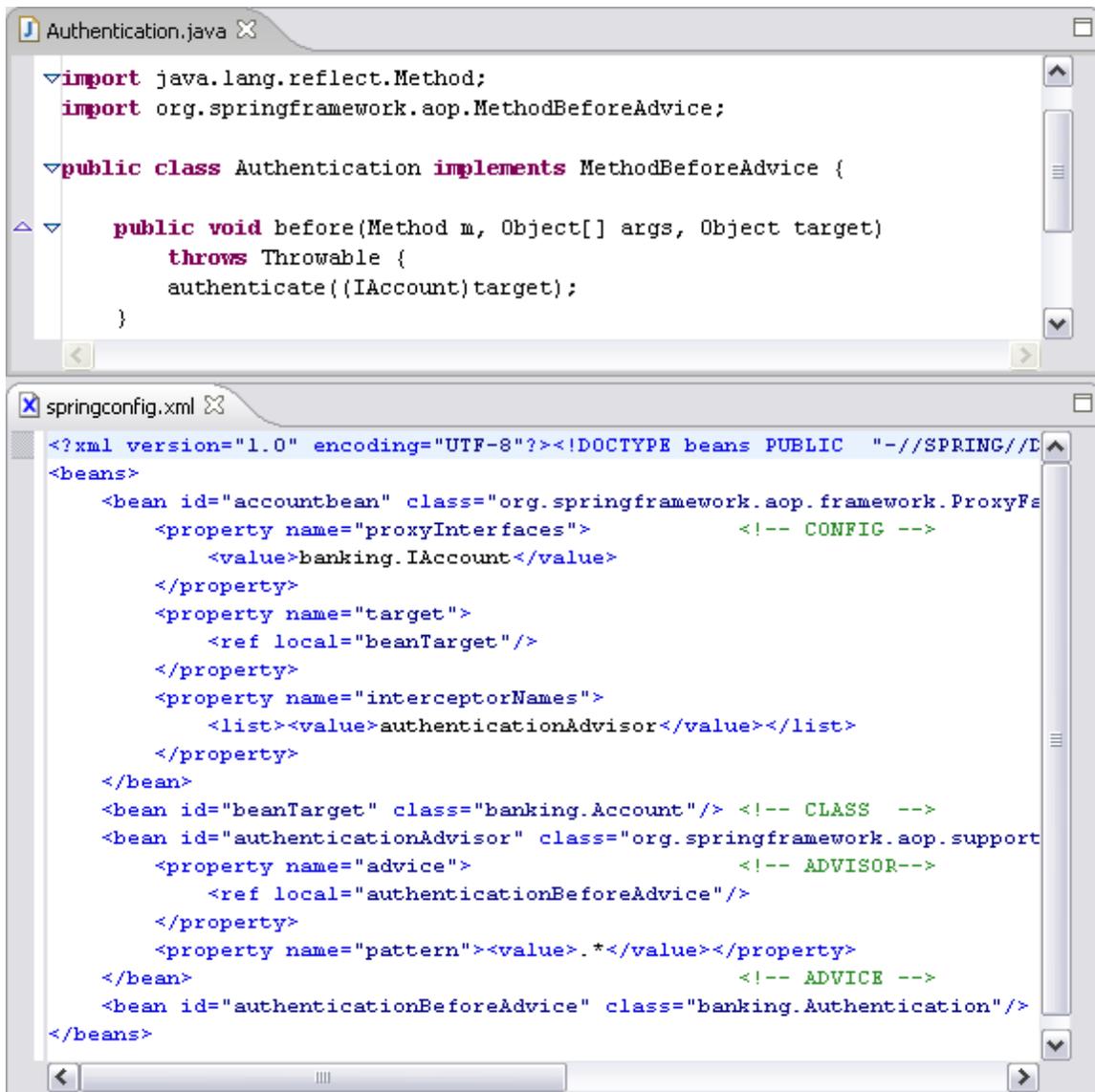


```
Authentication.aj
import banking.*;

public aspect Authentication {
    public pointcut authenticationRequired(Account account):
        execution(public * Account.*(..) && this(account));

    before(Account account): authenticationRequired(account) {
        authenticate(account);
    }
}
```

Fig. 2. An Aspect for papering unhandled exception in AspectJ [7]



```
Authentication.java
import java.lang.reflect.Method;
import org.springframework.aop.MethodBeforeAdvice;

public class Authentication implements MethodBeforeAdvice {
    public void before(Method m, Object[] args, Object target)
        throws Throwable {
        authenticate((IAccount) target);
    }
}

springconfig.xml
<?xml version="1.0" encoding="UTF-8"?><!DOCTYPE beans PUBLIC "-//SPRING//D
<beans>
  <bean id="accountbean" class="org.springframework.aop.framework.ProxyFe
    <property name="proxyInterfaces"> <!-- CONFIG -->
      <value>banking.IAccount</value>
    </property>
    <property name="target">
      <ref local="beanTarget"/>
    </property>
    <property name="interceptorNames">
      <list><value>authenticationAdvisor</value></list>
    </property>
  </bean>
  <bean id="beanTarget" class="banking.Account"/> <!-- CLASS -->
  <bean id="authenticationAdvisor" class="org.springframework.aop.support
    <property name="advice"> <!-- ADVISOR-->
      <ref local="authenticationBeforeAdvice"/>
    </property>
    <property name="pattern"><value>.*</value></property>
  </bean> <!-- ADVISOR -->
  <bean id="authenticationBeforeAdvice" class="banking.Authentication"/>
</beans>
```

Fig. 3. An Aspect for papering unhandled exception in Spring AOP [7].

IV. EVALUATION APPROACHES

Due to the potential benefits of AOP in software engineering and the tremendous advantages claimed by its supports, many studies have emerged to systematically evaluate the AOP approach and compare it to conventional programming approaches.

Ali et al. [10] have made a systemic review of comparative evidence of aspect-oriented programming. They discussed, in detail, the benefits and limitations of AOP based on the following criteria: performance, code size, modularity, evolvability, cognition and language mechanism. Each criterion was studied and was concluded with one of four possible results:

- Positive – when they note enhancement of the criterion with AOP compared to non-AOP implementations.
- Negative – when the implications of introducing aspects are not advantageous in the context.
- Insignificant – when AOP solution does not produce better results than earlier solutions, or there is no noteworthy evidence of enhancement.
- Mixed – when the study concludes with a combination of above three statement types and does not deliver any aggregated statement about the effect that AOP had on the studied characteristic.

The outcomes after evaluation each criteria are as follows:

- Performance: The results were Mixed results having AOP generating positive outcomes in regards to execution performance by improved response time and minimising the usage of both memory and hardware costs. However, the results were Insignificant when AOP was tested in Unix OS to evaluate runtime cost. The result of using AOP for optimising a network simulator was the same. This outcome made some researchers question if AOP can influence the performance.
- Code size: From the beginning, the founder of AOP, Kiczales [1], promised that his approach would create a tangible reduction in the size of code because of the separation of crosscutting concerns as mentioned in earlier sections. According to the research finding in this matter, there was a notable reduction in code size by approximately 40%, which means that there was a reduction in the line of code (LoC) as well. In addition, there was a reduction in certain types of codes such as exception handling. However, in some particular cases, AOP did not remarkable affect the LoC numbers. This led to the conclusion that AOP is actually effective in minimising the code size positively most of the time. If not, it will be more or less the same as non-AOP approaches.
- Modularity: Modularity results were positive, especially in Separation of Concerns (SoC). However, there was a lack of evidence in some studies, which suggests the need of more research in this area.

- Evolvability: Evolvability means AOP's ability to adapt to the continuous change in the user requirements and operational environment. Results were positive for this matter.
- Cognition: The cognitive outcomes were measured through looking at the development time and understandability, which is the degree to which developers/evaluators understand a system or component. Obtained results were insignificant so three studies were reviewed but results are not encouraging
- Language mechanism: The way that AOP deals with the code is certainly different from traditional approaches. Exception handling was taken as an example and compared in both OOP and AOP approaches. Results found were positive.

To evaluate the effectiveness of AOP in separation of concerns, Tsang et al. [14] applied a code quality metrics suite, developed in [18] to compare between real systems developed based on AOP and OOP in terms of system properties. They used the amount of reduction in coupling and cohesion values of the CK metrics as performance measures. The results showed better modularity of AOP systems over OOP systems,

Madeyski and Szala [4] have also made an empirical study of the impact of AOP on software development efficiency and design quality. Although their study has an obvious weakness, which is the small sample size (three programmers, only one of which is using AOP while the other two used OOP), it does give some research background for future studies. They asked the programmers to develop a web-based application for manuscript submission and reviewing. The goals of the study include:

- Evaluating the AOP impact on code quality.
- Evaluating the AOP impact on software development efficiency.

The researchers concluded their study by stating that the impact of AOP in software development efficiency was not confirmed. This is firstly because of the disability of applying statistical tests to analyze it due to the limited number of participants, as mentioned earlier. Secondly, it is because the statistical tests that they could execute for internal metrics showed insignificant results. That was also the case for the AOP impact on code quality: according to the researcher, the only positive impact in code quality was modularity.

Recently, Boticki et al. [2] investigated the educational benefits of introducing AOP paradigm into programming courses for undergraduates software engineering students. The study discusses how using the AOP paradigm, affects students' programs, their exam results, and their overall perception of the theoretically claimed benefits of AOP. The research methodology consisted of analyzing of students' programs, administering surveys, and collecting exam results. The results showed that the use of AOP as a supplement to object-oriented programming enhances the productivity of the students and leads to increased understanding of theoretical concepts.

V. CHALLENGES

So far, AOP has not gained wide adoptions. In addition to the possible reason related to it still being in infancy stage, some other disadvantages and challenges associated with it were highlighted in the following studies.

According to Laddad [9] there are two common oppositions to AOP, the first being that it makes the debugging process much harder. The second opposition is the fact that crosscutting modules implementation requires understanding the core module implementation details and vice versa. This is not the case in the OOP approach, though, where understanding is only required of the exposed abstraction between two classes. Moreover, Luca and Depsi [11] have discussed the challenges that AOP faces as a new programming approach in the following points:

- Lack of expertise: The community members of AOP are approximately only 2000 programmers worldwide, and only 10-15% of them are experienced enough to use AOP in an OOP environment.
- Concerns: Although AOP came to provide and to deliver a better separation of concerns (SoC), in reality, when a system reaches a certain degree of complexity, such separation is very hard to achieve, if not impossible.
- Standardisation: AOP introduced new dimensions and standards to programming. This, in general, creates complexity and possible resistance, but it was also the case when the OOP was introduced after the POP, which indicates that this is a normal scenario.

VI. CONCLUSION

AOP is a programming approach that aims to solve crosscutting concerns by offering better modularization of the code. This paper provided a brief overview of the state-of-the-art in AOP, starting with its definitions and example usages. It then went on to highlight the needs that led to the introduction of AOP. These can be summarized as the desperate demand for improved software quality. After that, an overview of previous works that have conducted evaluation studies of AOP were presented. The studies discussed the benefits and limitations of AOP based on performance, code size, modularity, evolvability, cognition, language mechanism and efficiency.

However, obtained results could not prove or disprove the effectiveness of AOP, except in two measures: language mechanisms and code size. AOP showed positive outcomes in these two measures. Possible threats and challenges associated with AOP were also discussed. They included making the debugging process harder and requiring more understanding of the core module and crosscutting concerns implementation. All these issues were not presented in conventional programming approaches.

All of the referenced research had a common conclusion, declaring the need of further in-depth studies and more research of AOP and its impact, which shows that this approach is still relatively new and unpopular. However, the developers who used this approach feel very confident and they

talk assertively about its enrichment to software quality. The empirical studies, though, had another thing to say, and it was not always in favor of AOP.

To conclude, it has been found that AOP is a very interesting topic that needs to take its righteous place in the programming community. Only then could researchers study AOP effectively and efficiently.

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), Finland, Springer Verlag LNCS 1241, June 1997.
- [2] 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.
- [3] 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.
- [4] L. Madeyski, L. Szala, "Impact of aspect-oriented programming on software development efficiency and design quality: an empirical study," IET Software, 2007 , vol. 1, no.5, pp. 180-187.
- [5] J. Viegas, J. Vuas, "Can aspect-oriented programming lead to more reliable software?," IEEE Software, 2000, vol.17, no.6, pp. 19-21.
- [6] 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.
- [7] M. Kersten, AOP@Work: AOP tools comparison, Part 1, accessed [26/8/2013] [online] available: <http://www.ibm.com/developerworks/library/j-aopwork1/>
- [8] H. Bing, C. Jiaying G. Jianye, "An Approach to Implement AOP Framework Under .NET Platform, Journal of Computer and Modernization," 2009, vol.11.
- [9] R. Laddad, "Aspect-oriented programming will improve quality," IEEE Software, 2003, vol.20, no. 6, pp. 90-91.
- [10] 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.
- [11] L. Luca, I. Despi, "Aspect Oriented Programming Challenges," Anale Seria Informatica, 2005.vol. 2, no. 1, pp. 65-70.
- [12] R. D. Dechow (2005), "Advanced Separation of Concerns and the Compatibility of Aspect- Orientation. accessed [26/8/2013] [online]
- [13] [http://ir.library.oregonstate.edu/xmlui/bitstream/handle/1957/29757/Dec howDouglasR2005.pdf?sequence=1](http://ir.library.oregonstate.edu/xmlui/bitstream/handle/1957/29757/Dec%20DouglasR2005.pdf?sequence=1)
- [14] G.Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, W. G. Griswold, "Getting Started with AspectJ", CACM, 2001, vol. 44, no. 10.
- [15] S.L. Tsang, S. Clarke, E.L.A. Baniassad, "An evaluation of aspect-oriented programming for Java-based real-time systems development," In Proceedings of the International Symposium of Object-Oriented Real-Time Distributed Computing ISORC, 2004, Vienna, Austria, pp. 291-300.
- [16] K. Lieberherr, D. Orleans, J. Ovlinger, "Aspect-oriented programming with adaptive methods," ACM Communication, 2001, vol. 44, no. 10, pp 39-41.
- [17] T. Xie, J. Zhao, "A framework and tool supports for generating test inputs of AspectJ programs," In Proceedings of AOSD, 2006, pp. 190-201,
- [18] A. Colyer , A. Clement , G. Harley, M. Webster, Eclipse AspectJ: Aspect-Oriented Programming With AspectJ and the Eclipse AspectJ Development Tools. Addison-Wesley pp.504, 2004.
- [19] 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.

- [20] E. M. Novikov, "An approach to implementation of aspect-oriented programming for C," *Programming and Computer Software*, 2013, vol.39, no. 4, pp 194-206.
- [21] E. M. Novikov, "One approach to aspect-oriented programming implementation for the C programming language, Proc. of the 5th Spring/Summer Young Researchers' Colloquium on Software Engineering, Yekaterinburg, 2011, pp. 74-81.
- [22] 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.
- [23] AspectJ: an aspect-oriented extension to Java. <http://www.eclipse.org/aspectj/doc/released/progguide/language.html>
- [24] Introduction to AspectJ. <http://eclipse.org/aspectj/doc/released/progguide/starting-aspectj.html>