CHECKING CONSISTENCY IN TELECOM OPERATION USING AOP

[1]Sk. Riazur Raheman,[2] Hima Bindu M, [3]Amiya Kumar Rath
[1]Ph.D Scholar, [2]Professor, [3] Professor
[1]Department of Computer Science & IT, [2]Department of Computer Application, , [3]Department of Computer Science & Engineering
[1]North Orisa University, Baripada, India,,[2] North Orisa University, Baripada, India, [3]VSSUT, Burla, Odisha, India,

Abstract: Aspect oriented programming brings the paradigm shift in the approach of software development. Using aspect oriented programs crosscutting concerns are handled very effectively. To enable the consistency checking in a telecom operation it is better to have the different concerns in a separate module. This can be achieved easily using the concept of aspect oriented programming. The application of aspect oriented programming affects multiple concerns of telecom operation and hence is a crosscutting concern. The benefit of aspect oriented programming is its advanced modularization capabilities which are capable of modularizing crosscutting concerns. The paper discusses the development of aspect oriented application for the consistency check of telecom operation. The crosscutting concerns in telecom operations are encoded in aspect modules for implementation. Class and sequence diagrams are designed to depict the telecom operation.

Keywords: Aspect, Aspect oriented programming, Crosscutting, Consistency check, Design.

I. INTRODUCTION

The complexity and the requirement of software development have been increased remarkably in the past. The reason for this development is because of the demand of more sophisticated software. To fulfill this demand the development of software are going to be in modular approach. These modular approaches are handled in different programming approaches differently. However, there is a possibility of these modules may overlap with each other. The concern which exists in different modules is called as crosscutting concerns. This means that crosscutting concerns are scattered in different modules. Also these crosscutting concerns are tangled with the code of other concerns within a module. If the concerns are scattered in different modules then it reduces the quality of software. These crosscutting concerns cannot be generalized as a separate concern in object oriented programming. Aspect oriented program works as a better tool for handing this crosscutting concern. Thus by using the concept of aspect oriented program, it helps in improving the quality and modularity of a software.

The performance concerns vital to telecom consistency check cannot be effectively handled in object oriented programming. The concerns associated to consistency checking crosscut the basic functionality. We need a good programming style that allows developing the crosscutting concerns as a separate module without disturbing other basic functionality. There are different types of crosscutting concerns exist in telecom operations. The concerns like timing, billing and consistency check are crosscutting concerns. Consistency check in a telecom operation is to make sure that all calls and connections are being shut down in the reality.

II. ASPECT ORIENTED PROGRAM AND CROSSCUTTING CONCERNS

Aspect Oriented Programming (AOP) is an encouraging new knowledge for separating crosscutting concerns. Gregor Kiczales [1] was the first to present the concept of AOP. He worked on AOP at the Xerox Palo Alto Research Center (PARC) and stood as a leader in the development of AOP. AOP improves the system features by adding modularity, understanding and ease by better handling of crosscutting concerns. To modularize and localize the code from crosscutting concerns, AOP uses aspects as a special module. Some of the prominent AOP features are point-cut, join point and advice. A point-cut defines a set of join points in an AOP. Join point used to link the aspect and non-aspect code. An Advice is associated with a point-cut, it is used to implement the crosscutting functionality. There are three types of advices in an AOP, before, after, and around. AOP addresses the difficulties triggered by crosscutting concerns in the following means [2, 3, 9, 11, 12, 24].

- Decomposition of the software into modules which holds the concerns that do not crosscut. This module is known as non-aspect code. This can be written by simple java code.
- Identification of the concerns that crosscut the operation of other concerns and capturing and applying these concerns into a separate module known as aspect. This module is known as aspect code.

Consider the Banking System (BS) to open an account; different phases are registration, verification of identity and address of a customer. The concerns related to accounts include minimum balance checking, withdraw and deposit. All these requirements are core concerns of the banking system. Also a banking system must have some requirements related to security and recovery to ensure the data is well maintained. These additional requirements are crosscutting concerns, because these concerns have an impact on other core concerns. In Figure 1, all the core concerns are shown as vertical columns and the crosscutting concerns are shown as horizontal bars that cross the vertical columns. The crosscutting concerns are the main point of interest of an AOP.

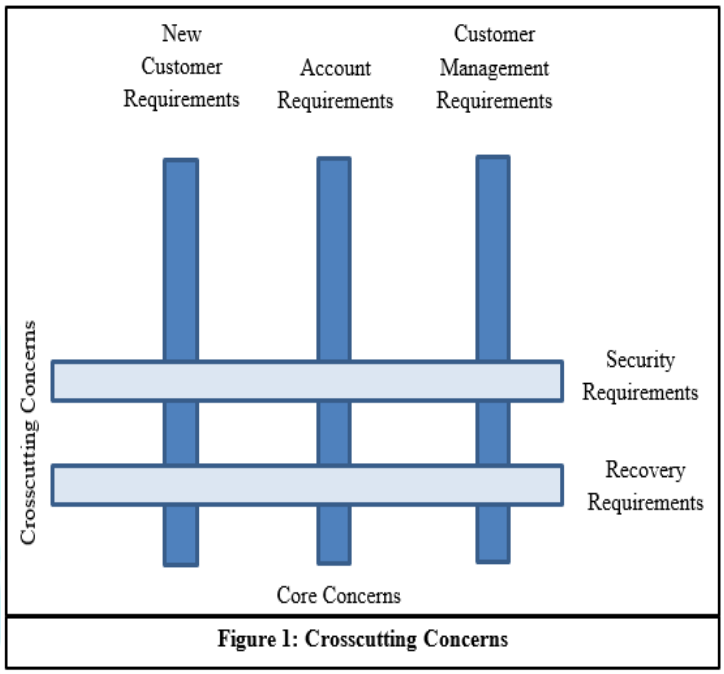

Figure 1: Crosscutting Concerns

AspectJ is the best standard tool available for AOP development. AspectJ is an implementation of AOP. AspectJ is developed by Xerox Palo Alto Research Center. It is similar to java and practices similar java like syntax. AspectJ based on the object model of Java by means of some extra features that allow AOP procedures to be used. Core concerns are pure Java code and crosscutting concerns are aspect code. Some mapping or weaving rules are implemented to link the crosscutting concerns with primary concerns. An AOP program is compiled into byte code, and can run on any Java platform. AOP is written in two parts.

1. **Base code**
2. **Aspect code**

Base code contains primary concerns like classes and other standard Java concepts whereas **aspect code** includes crosscutting concerns.

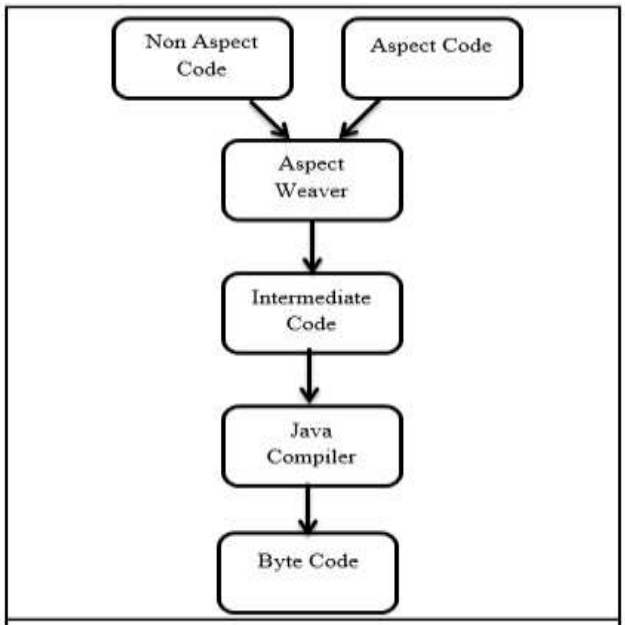

Figure 2: Aspect Weaving Process

In an AOP, core concerns (non-crosscutting concerns) and aspects (crosscutting concerns) are developed independently [9, 12, 13]. Both the aspect and non-aspect code are joined into an ultimate executable form using the aspect weaver. As a result, a single aspect code can help in the operation of total modules which increases both reusability and maintainability of the code [6, 7, 9, 10]. The process of aspect weaving is depicted in Figure 2.

III. ASPECT ORIENTED DESIGN APPROACH

Aspect oriented system designing includes some activities. While developing an aspect oriented system, all the activities must be linked properly [26]. The activities are given as follows:

1. Core system concern
2. Aspect identification and design
3. Composition design
4. Conflict analysis and resolution
5. Name design

Core system design: Initially, system architecture is designed based on the core concerns of the system. Also the dependability among the different concerns is addressed in this stage.

Aspect identification and design: This is the phase to identify different aspects in the system. Once the aspects are recognized, these can then be developed independently based on the design of the core system features.

Composition design: At this stage, aspects are linked with the core system. The core system and aspect are analyzed to find out the point of linking the aspect with the core system. This is also called as finding the join points for aspect woven.

Conflict analysis and resolution: At the time of composition design, concerns related to aspect and point-cuts must be handled properly. Otherwise, this may lead to weak and unwanted software. Linking between aspect and non-aspect code should be taken care properly. A fault point-cut definition or a wrong join point may lead to unpredictable output. Advices implementing the crosscutting concerns must be in proper sequence.

Name design: This deals with defining principles for identification of different entities in the program. It helps in avoiding accidental point-cut and wrong join points. To overcome any unpredicted program behavior, we should plan a naming system that reduces the probability of this happening.

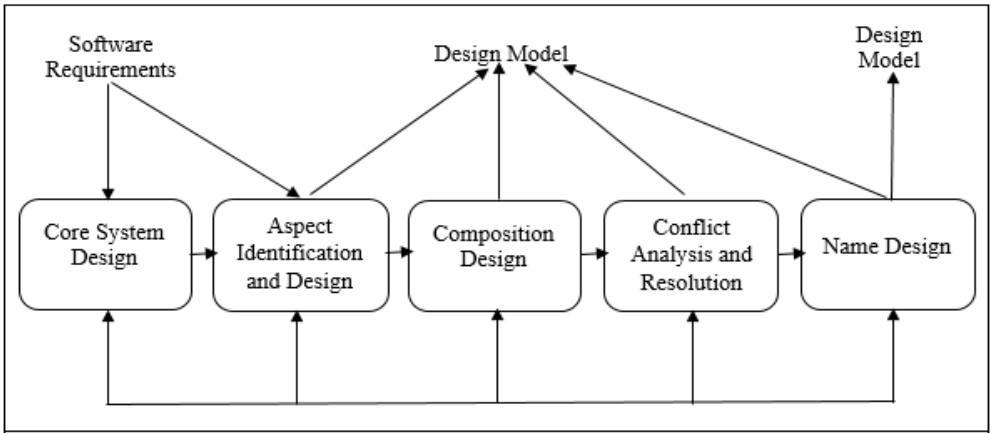

Figure 3: Aspect Oriented Design Process

Figure 3 represents the different activities during the aspect oriented system design and the relations among them.

IV. BASIC DESIGN APPROACH FOR TELECOM OPERATION

In this section we will discuss about the design approach of a telephone operation. The class diagram, sequence diagram and consistency check in a telephone operation will be discussed.

4.1 Class Diagram

In object oriented system development, the class diagram is the foundation for the system model. A class diagram gives a picture of the relationship between different classes. One can understand the working of each class by viewing at the related functions. The multiplicity is also shown to know the system better. In our design approach we have considered three classes as call, connection and customer [9, 14, 15, 23, 24]. Also we have added the different functionality to each class as shown in Figure 4. The methods for handling the calls are described in Customer class. The Connection class represents the physical particulars of making a connection among customers. The Call class is shaped for both caller and receiver. If the area code of caller and receiver are same then the call is treated as Local connection. Otherwise a LongDistance connection is required [19, 25].

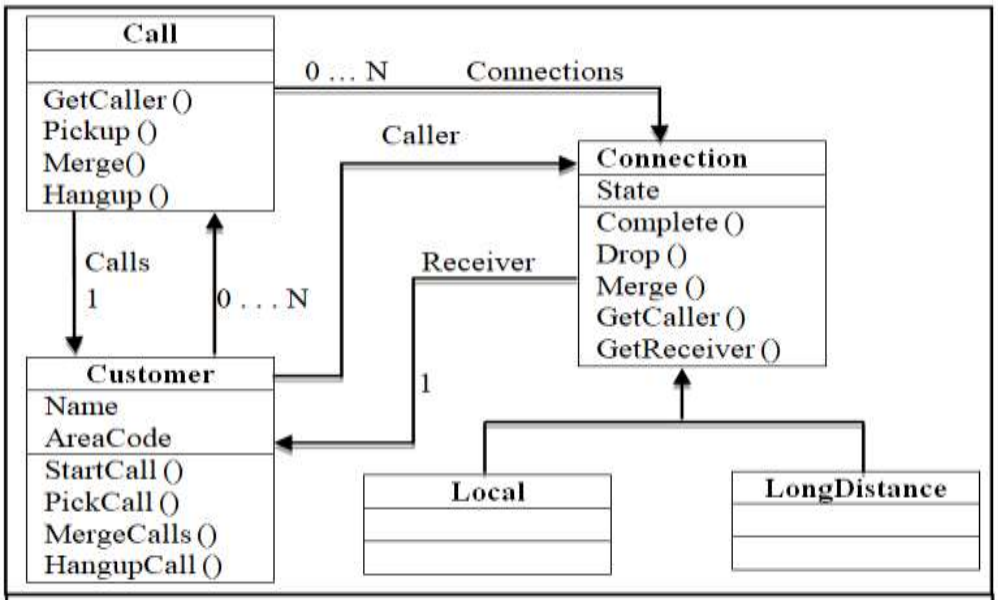

Figure 4: Design approach for telecom operation

4.2 Sequence Diagram

A sequence diagram demonstrates a communication between different objects organized in time sequence and the messages that pass among them when an interaction takes place. In an aspect oriented software design phase, it is significant to define the correlation between point-cut and aspects [8, 14, 15]. This can be represented effectively by a sequence diagram as shown in Figure 5.

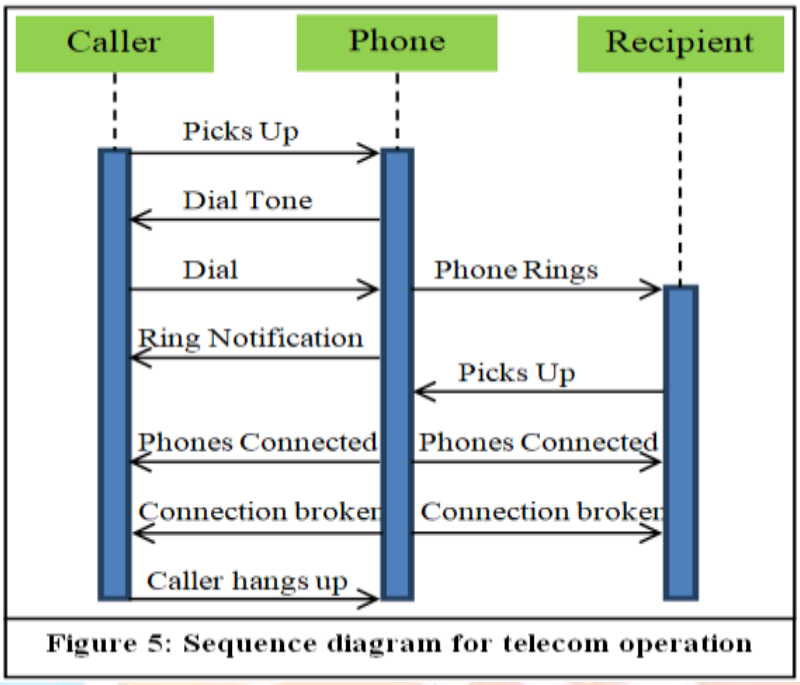

Figure 5: Sequence diagram for telecom operation

In our design concept we are considering the major objects as caller, phone and recipient. The communication between objects is built on operation calls mechanism. The system mechanism is, when a caller picks up his phone, the network reacts by sending a tone. This user is now prepared to dial the telephone number of the recipient. Then the network directs back a signal which effects a ring on the called phone. An Echo ring is then sent to the caller. We undertake that the called user is constantly ready to answer a call. When the called user picks up his phone, the ring is then interrupted and the two users involve in a talk [19, 20, 25].

4.3 Consistency Checking

The object oriented programming approaches lack some in-built method calls which are necessary before, after or during an event occur to handle the crosscutting concern, whereas in AOP, it has an approach like aspect to handle crosscutting concerns. The performance concerns vital to telecom consistency check cannot be effectively handled in object oriented programming. The concerns associated to consistency checking crosscut the basic functionality. We need a programming style that allows developing the crosscutting concerns as a separate module without disturbing other basic functionality. Also it should handle the performance issues efficiently even though they crosscut the module structure of the basic functionality. AOP is mainly useful for this type of challenges [4, 16, 22, 23].

Consistency check in a telecom operation is to make sure that all calls and connections are being shut down in the reality [3]. There are different types of crosscutting concerns exist in telecom operations. The concerns like timing, billing and consistency check are crosscutting concerns. The phone connection time is handled by timing aspect, the billing aspect uses the connection time to bill the originator of the call and the consistency check aspect checks the hang-up and merge calls [5, 17,21].

The consistency check aspect has at most interfaces with the methods in the program, leaving its performance complete within its original scopes. Consistency check handles the concerns related to hang-up, merge and drop connections [5]. The concerns related to consistency check are depicted in Figure 6.

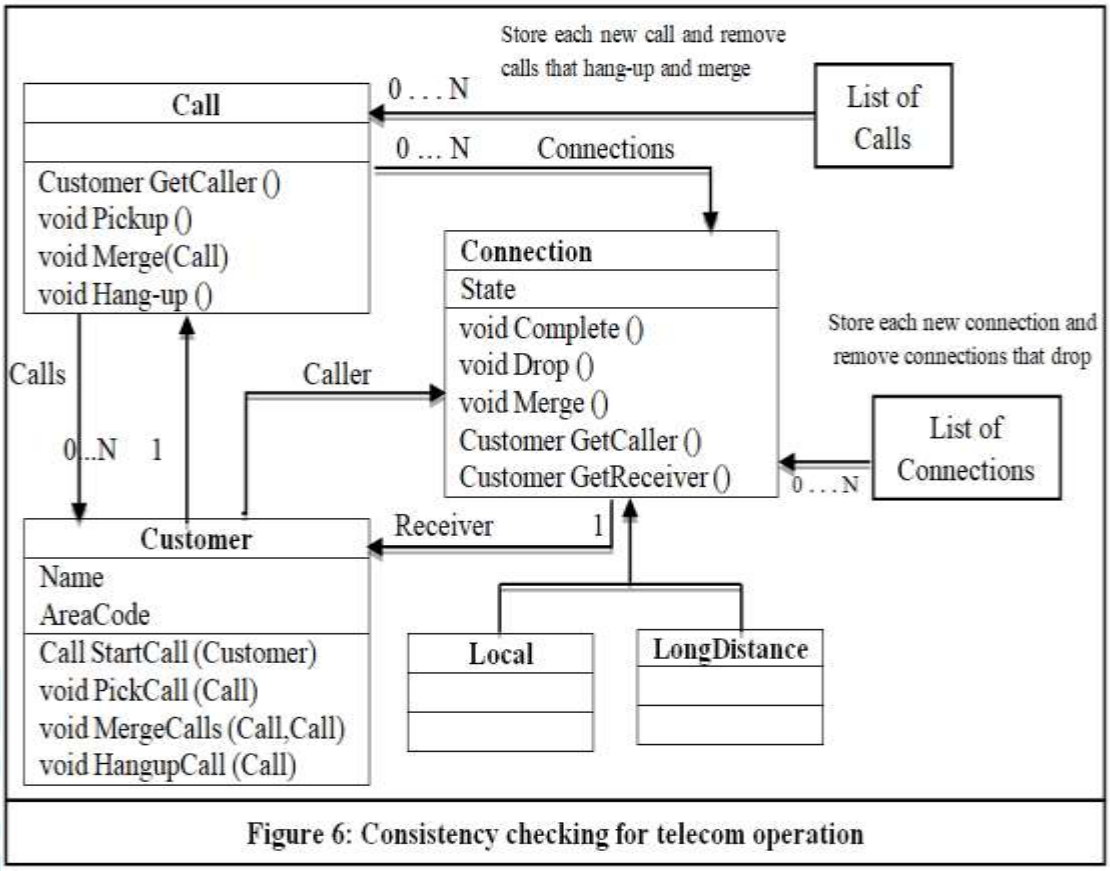

Figure 6: Consistency checking for telecom operation

V. DESIGNING ASPECT FOR CONSISTENCY CHECKING

In a telephonic call, call waiting and the conference call are two features which must be handled properly for consistency check [20]. Call waiting feature allows a subscriber already engaged in a communication to be informed if a new user attempts to reach him. X can either overlook the new call or can get connected to the new call. Another feature is a conference call. It allows three users to interconnect in the following way: Consider a subscriber X who is communicating with Y. X can then add Z in the talk. To achieve this objective, X first put Y on hold by pressing hold button. Then, start a communication with Z. Finally, press the hold button again, to get, X, Y and Z are connected. X can remove Z from the conversation by pressing the button. If X hangs up, Y and Z remain in communication [20].

In order to demonstrate the role of aspects in consistency checking in telecom operation, an aspects has been designed based on different functions. The major units for aspect design are as follows:

1. Caller
2. Phone
   a. An after advice to add new calls
   b. An after advice to remove calls when recipient hang-up
   c. An after advice to remove calls when a call merge with other calls
   d. An after advice after connection established
   e. An after advice after connection drops
3. Recipient

A customer pickup call, merge calls and hang-up calls. Like that the functionality of call module is to store each new call and removes the call that hang-up and merge. The functionality of connection module is to store each new connection and remove connections that drop [16, 17, 18]. The concerns related to call and connection is depicted in the Figure 7, using a simple Java code.

```
Call
{
        Customer _caller;
        Vector _connections = new Vector();

        Customer getCaller()
        {
                return _caller;
        }

        Call(Customer caller, Customer receiver)
        {
                _caller = caller;
                Connection conn = null;
                if (receiver.areacode == caller.areacode)
                conn = new Local(caller, receiver);
                else
                        conn = new LongDistance(caller, receiver);
                        _connections.addElement(conn);
        }

        void pickup()
        {
                ((Connection)_connections.lastElement()).complete
        ();
        }
}
```

Figure 7: Java code for call and connection concerns

All these crosscutting concerns are depicted in aspect oriented design of consistency checking in Figure 8. In the aspect described in Figure 8, consists of two point-cuts one for calls and another one for connections. It creates two vectors as, calls and connections for updating the calls and connections established. The point-cut described for calls, handles three situations like, new call, hang-up and conference call. When a new call established then that will be added to calls vector, it is shown in the first after advice. When the connection hang-up then that call is removed from the calls vector as shown in the second after advice. In the third after advice of calls point-cut, it is showing that, when there is a conference call to other (X), and then other (X) will be removed from the calls vector.

```
aspect ConsistencyChecker
{
        Vector calls = new Vector(), connections = new Vector();

        /*The lifecycle of calls */
        after(Call c): receptions(c.new(..)) {
                calls.addElement(c);
        }
        after(Call c): receptions(* c.hangup(..)) {
                calls.removeElement(c);
        }
        after(Call other): receptions(void Call.merge(other)) {
                calls.removeElement(other);
        }

        /* The lifecycle of connections */
        after(Connection c): receptions(c.new(..)) {
                connections.addElement(c);
        }
        after(Connection c): receptions(* c.drop(..)) {
                connections.removeElement(c);
        }
        after(): within(TelecomDemo) && executions(void main(..)) {
                if (calls.size() != 0) println("ERROR on calls clean up.");
                if (connections.size()!=0) println("ERROR on connections clean up.");
        }
}
```

Figure 8: Aspect for consistency checking

Like that the point-cut described for the connection addresses three concerns like, new connection, drop connection and for handling error. In the first after advice of connection, when a connection established that one is added to connections vector. Once the connection is dropped it will be removed from the connections vector that is shown in second after advice. The last after advice of connection point-cut checks the two vectors, calls and connections at last. If any of the vectors is showing any nonzero value, then there may be some error in connections clean up. So the aspect defined in Figure 7 describes the details about the calls and connections.

VI. CONCLUSION

In this paper, we suggested a formal approach to detect and resolve feature interactions within a telecom operation. As handling crosscutting aspect thrown a lot of challenges, the proposed approach tried to solve the crosscutting concerns arises in a consistency check of telecom operation. We have explained how AOP can help in minimizing the code complexity of systems without losing essential performance necessities. We have designed the class and sequence diagrams based on crosscutting concerns. Also we have implemented an aspect on consistency check.

VII. ACKNOWLEDGMENT

We would like to express our special thanks of gratitude to Raajdhani Engineering College,Bhubaneswar to motivate us to do research and development works on Arduino based Embedded system.

References

[1] Kurdi, Heba A. 2013. "Review on Aspect Oriented Programming." (IJACSA) International Journal of Advanced Computer Science and Applications 4, no. 9: 22-27.

[2] Griswold, Bill et. al. 2001. "Aspect-oriented programming with AspectJ." AspectJ. org, Xerox PARC.

[3] Raheman, Sk Riazur et. al. 2011. "Dynamic slicing of aspect-oriented programs using aodg." International Journal of Computer Science and Information Security 9, no. 4: 123.

[4] Laukkanen, Jyri. 2008. "Aspect-Oriented Programming." University of Helsinki, Department of Computer Science.

[5] Pollice, Gary. 2004. "A look at aspect-oriented programming." Online article: http://www. ibm. com/developerworks/rational/library/2782. html Published Feb 17.

[6] Tapan Kant et. al. 2015. "Redesign of Hot Spots using Aspect-Oriented Programming". International Journal of Computer Applications (0975 – 8887) Volume 117 – No. 19, May.

[7] Kiczales, Gregor et. al. 2001. "An overview of AspectJ." ECOOP 2001—Object-Oriented Programming : 327-354.

[8] Robinson, David. 2006. "An introduction to aspect oriented programming in e." 2006-09-15]. http://www, verilab. com/-downloads, html.

[9] Qamar, M. N., Aziz Nadeem, and R. Aziz. 2007. "An Approach to Test Aspect-oriented Programs." In World Congress on Engineering, pp. 211-216.

[10] Vaira, Žilvinas, and Albertas Čaplinskas. 2011. "Case Study Towards Implementation of Pure Aspect-oriented Factory Method Design Pattern.".

[11] Raheman, Sk Riazur et. al. 2013. "An overview of program slicing and its different approaches." International Journal of Advanced Research in Computer Science and Software Engineering 3, no. 11: 435-442.

[12] Berardi, Daniela et. al. 2005. "Reasoning on UML class diagrams." Artificial Intelligence 168, no. 1-2: 70-118.

[13] Eriksson et. al. 2003. UML 2 toolkit. Vol. 26. John Wiley & Sons.

[14] Jose M. Felix et. al.. 2014. "Aspect-Oriented Programming to Improve Modularity of Object-Oriented Applications". Journal of Software, VOL. 9, NO. 9, September.

[15] Rehab Allah Mohamed Ahmed et. al. 2017. "Extending Unified Modeling Language to Support Aspect-Oriented Software Development". In International Journal of Advanced Computer Science and Applications, Vol. 8, No. 1, pp. 208-215.

[16] Fatima Beltagui, et. al. 2013. "Features and Aspects: Exploring feature-oriented and aspect-oriented programming interactions". Technical Report No: COMP-003-2003, Computing Department, Lancaster University, Lancaster, May.

[17] Madadpour et. al. 2011. "Testing Aspect-Oriented Programs with UML Activity Diagrams." International Journal of Computer Applications33, no. 8 (2011).

[18] Tom Dinkelaker et. al. 2011. "Using Aspect-Oriented State Machines for Resolving Feature Interactions". In the Proceedings of the Federated Conference on Computer Science and Information Systems pp. 809–816, ISBN 978-83-60810-22-4.

[19] Mendhekar et. al. 1997. RG: A case-study for aspect-oriented programming. Vol. 9710044. Technical Report SPL97-009, (1997).

[20] Stein, Dominik et. al. 2002. "A UML-based aspect-oriented design notation for AspectJ." In Proceedings of the 1st international conference on Aspect-oriented software development, pp. 106-112. ACM, (2002).

[21] Yuliyan Kiryakov and John Galletly. 2003. "Aspect-Oriented Programming – Case Study Experiences". International Conference on Computer Systems and Technologies – CompSysTech. (2003).

[22] Rinard, Martin et. al. 2004. "A classification system and analysis for aspect-oriented programs." In ACM SIGSOFT Software Engineering Notes, vol. 29, no. 6, pp. 147-158. ACM.

[23] Stein, Dominik et. al. 2002. "Designing aspect-oriented crosscutting in UML." In Workshop on Aspect-Oriented Modeling with the UML, AOSD'02.

[24] Lidia Fuentes et. al. 2007. "Designing and Weaving Aspect-Oriented Executable UML models". In the Journal of Object Technology. Vol. 6, No. 7, Special Issue: Aspect-Oriented Modeling, August (2007).

[25] Groher, Iris, and Stefan Schulze. 2003. "Generating aspect code from UML models." In The 4th AOSD Modeling With UML Workshop.