Sindh Univ. Res. Jour. (Sci. Ser.) Vol. 47 (4) 775-778 (2015)

SINDH UNIVERSITY RESEARCH JOURNAL (SCIENCE SERIES)

Cross-cutting Concerns of Modularization in Object-Oriented and Aspect-Oriented Paradigms

I. A. SUPRO, J. A. MAHAR[++], J. A. MANGNEJO, M. A. ANSARI*

Department of Computer Science, Shah Abdul Latif University, Khairpur Mir's, Pakistan

Received 09th December 2014 and Revised 12th August 2015

Abstract: Software development process contains of many features and modularization is one of the core features. Modularization is the process to divide a program into different modules and each module individually performs certain task. This methodology is very useful in software design and works very well but in various situations particularly in single class this method does not work properly. Synchronization and logging are cross-cutting concerns that are parts of program which interrupt the other components of program and generate different possibilities for code tangling and code scattering. Therefore, maintainability and readability of software may also be affected. Cross-cutting concerns are supported by Object-Oriented partially and fully by Aspect-Oriented. Comparative study of modeling and coding issue of cross-cutting concerned in Object-Oriented and Aspect-Oriented paradigms are presented in this paper. Development of more customizable software is assured by Aspect-Oriented using encapsulated aspect to knob the cross-cutting concerns. It has been analyzed that Aspect-Oriented is less error prone and it is much easier to debug and maintain the program than the Object-Oriented.

Keywords: Cross-Cutting, Modularization, Aspects, Object-Oriented, Aspect-Oriented

1. INTRODUCTION

The paradigm of programming is a method which serves as a school of thoughts for programming of computers. These paradigms are not limited to that but also reflect the different aspects of language designers. Most of the practical languages represent the features of more than one paradigm like java. Most familiar from which the programming paradigms are, imperative, functional, logical, object-oriented, Aspect-Oriented, Visual, Parallel and constraint based. This paper consisting two major paradigms i.e. Object-Oriented (OO), Aspect-Oriented (AO) are enumerated to compare the utilization and the effectiveness of modularity. The OO paradigm is used in various computational applications, for instance (Udayakumar, 2013) used this paradigm in distributed systems. On the other hand, AO paradigm is recently proposed which supports the modularization at maximum level through the aspects. More research studies are yet required to refactoring the developed applications (Subramaniam, 2014).

The idea of modularity in software development is taking up for about last six decades. The software design represents the modularity; which explains that software is divided into different components as per the requirement of the user or system, mostly called modules which further integrated to satisfy the problem requirement (Pressman, 2001). The basic idea of Simula, Smalltak, C++ and Java languages was to offer the modularity.

While developing software, the separation of concerns is a major subject matter. For the sake of this, various programming paradigms have been proposed. It is believed that the OO paradigm is one of the most widely used paradigms. As in OO paradigm, some concerns named crosscutting concerns are very difficult to modularize and are mostly scattered over modules in a program (Mesbah, 2005). These demerits of OO paradigm became the reason for the creation of a new paradigm, the AO paradigm. As an additional supplement to OO paradigm, the AO paradigm is designed and proposed to facilitate the software developer in order to overcome the problem of cross-cutting concerns. The AO paradigm also modularizes the crosscutting concerns as aspects that merge the codes into other modules.

Bundle of research works contributed by the researchers published in the books and research papers regarding the modular approach of OO and AO programming. This paper represents many of these, for instance, (Asagba, 2008) discussed and distinguished structural programming with that of OO programming. Similarly for AO programming (Amirat, 2008) has discussed sequential systematic activities towards an early consideration of identifying and separating the cross-cutting and (Kamle, 2009) has discussed that these concerns appear to affect the multiple implementations of modules

2. MODULARIZATION PROBLEM

A program is composed of one or more modules which are not united until the program is linked. Modular programming is a software design technique which highlights the separating functionality of a program into independent, interchangeable modules, as each of which

[++]Corresponding author: Email: mahar.javed@gmail.com
* University of Sindh, Jmashoro

is equipped with necessary contents to perform only one aspect of the preferred functionality.

The problem to appropriately modularize the software is a central issue to be kept in mind while developing large software projects. Appropriate modularization includes code readability, restricts the appearance of bugs due to loss of control on program code complexity and increase the opportunity of initiating adaptive changes in the large software project.

For the sake of improving the modularization of large software projects, the OO paradigm was proposed and introduced a new mechanism. According to this paradigm software has to be modularized subsequent a functional division. Even with the success of OO in the work for obtaining separation of concerns, certain attributes in OO mechanism can't frankly designed from given task area to the implementation, consequently they are not able to localized in a module. Due to this fact it is said that the functional decomposition is performed with class (Amirat, 2008).

Functional decomposition usually works properly, but for specific concerns. Such kinds of functionalities are not limited in a class and thus implementing them following a functional decomposition, implies to extend the implementation on various classes. This obliges at least two indications on implementation, code scattering and code tangling.

3. CROSS-CUTTING CONCERNS

Cross-cutting concern is the set of required information of the user existing in a module or program affecting other components of the software system. It may be error handling, security and non-functional characteristics exist in program coding (Marin, 2007). Mostly, the concern arises in a program either in scattered or tangled manner (Yang, 2010). Tangling means the main logic of the program is mixed up with the concern. In the mean time, scattering refers to the situation where by the identical concern is used again and again in a program.

Consider, we have to create a mathematical application which can perform arithmetic operations. Java exception handling mechanism is suitable enough for such application. From the sample illustrated in (Fig. 1), the concern which tangled and scattered in this program is exception concern. The purpose of this concern is to book the exception if generated. Since it is not the major component of the program, it is considered as crosscutting concern which crosscuts within and between the methods.

One of the important rules of the design of modern software systems for industries as well as academia is the "separation of concern" because it affects the readability, understandability, maintainability and other attributes of the software. The separation of concerns permits to compact with the dissimilar aspects of a selected problem, as one can concentrate on each independently (Ghezzi, 2003).

Both of the selected paradigms support modular approach which deals with the major problems of concern separation. The OO approach does not completely support the cross-cutting concern and results loss of modularity. However, the AO approach completely supports cross-cutting concerns. In AO approach, each of the cross-cutting concern is implemented in a separate file, named aspect. Each aspect consists of some portions of suggestions which implement the specified functionality. Each suggestion is concerned with some points in main program code, named join points. During "Run Time" the program code of each suggestion is executed with those joint points which are concerned to reach at the particular place in the program successfully.

```
class Supro
{public static void division ()
{int n, m;
   try
{ n = 0;m = 17 / n;
 System.out.println("Is number divided by zero");}
catch (ArithmeticException e)
{System.out.println("Error: Don't divide a number by zero");}
 System.out.println("I'm out of try-catch block in Java.");}
public static void Checkdivision ()
{System.out.println("Is number divided by zero");}}
```

Fig.1 Sample of Tangling and Scattering

4. MODULARIZATION IN OBJECT-ORIENTED PARADIGM

Modularization is the concept to split the computer program into modules, connect and combine them to develop a complete software system. The basic rule of modularity is that; do not pack all the programming logics into just one program.

The OO paradigm is suitable enough to design and develop the large software projects, as modularity facilitates to break the development responsibilities between software developers team. The OO programming enabled the software development process rapid and reliable (Udayakumar, 2013). Moreover, modularity is the core programming feature for the process of creating classes. The classes can be modeled by implementing Encapsulation, Inheritance and Polymorphism. The main advantages of modularity are reusability, code replication, robustness and exception handling.

Inheritance is a basic component of OO paradigm in which classes are derived from the base class. The derived classes bring all the attributes, services, data structures and algorithm associated with its base class. Its reutilization is accomplished directly and modularity approach is used at maximum level for the development and implementation of different types of inheritance. There is some uneasiness between the use of inheritance for abstract classes and distribution process during implementations. Most of this uneasiness is associated with the overriding methods. The problem appears when the overriding method significantly differs from the overridden method (Rumbaugh, 1999).

An inheritance model for University members applying overriding methods is represented in **(Fig. 2).** Programming code for this model specifically two functions getdata() and putdata() imposes the indications of tangling and scattering because both of the functions are present in base class as well as in derived classes. The object of class graduate preliminary calls getdata () of own class but as graduate class is derived from the student class and this class is derived from University member's class. Therefore, for obtaining complete data of all attributes, getdata() must call in every class hierarchy by utilizing the scope resolution operator. Moreover, overriding process is executed for the reasons: overriding for extension, restriction, optimization and convenience.

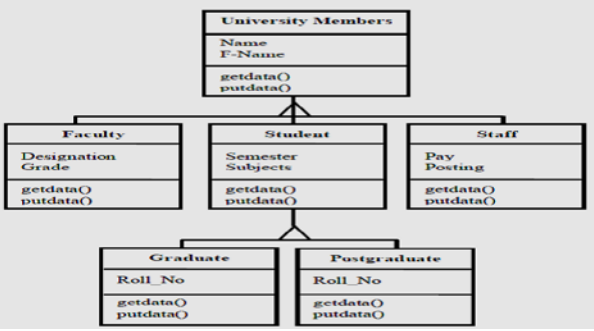

Fig.2 Inheritance Model of University Members

5. MODULARIZATION IN ASPECT-ORIENTED PARADIGM

Paradigms are constantly proposed by scientific researchers to fulfill the needs of end-users. The AO paradigm is comparatively a new one which attractively invites the software developers for their software applications. The general architecture of the AO paradigm is illustrated in **(Fig.3).** Though, OO is cost-efficient, a rapid way to design a good quality software and cuts development time but AO paradigm enables programming logics easier and faster, closer to human perception, makes programming less error-prone and easier to debug and maintain (Papapetrou, 2004). It also develops less expensive applications, software which is flexible and more customizable.

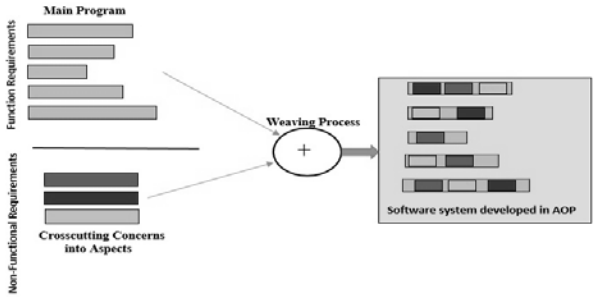

Fig.3 Architecture of Aspect-Oriented Paradigm

AO programming complements OO programming by providing another suitable and effective option about the program structure. In OO programming, the class is a key and basic unit of modularity, whereas in AO programming the unit of modularity is the aspect and it establishes a straight deal with aspects of concern rather than software modules.

Basically, aspect is a piece of code, which encapsulates the behaviors affecting different multiple classes into reusable modules. The aspects give modularity of cross-cutting concerns at a single place. So that software developers have become able enough to think, write, edit and address such type of issues as a module and coding can change or upgrade in all the associated sections of program, in spite of changes made in piece of code.

Let suppose the banking application for managing the automatic online fund transfers. For this we will have to design the classes of customer, teller and account. Only valid individual can transfer the funds by withdrawing the amount, while rest of the other users must be blocked and cannot be allowed to entertain any fund transfer request, because they are invalid users at that time. Possibly we must create and design the separate classes for customer and teller, adding separate codes which authenticate the user; the code which displays the authenticating concern is depicted in **(Fig.4).** This type of behavior causes the scattered code problem. Moreover, when the completed functionality is called and added into whole system, we required the separate calls of these scattered modules.

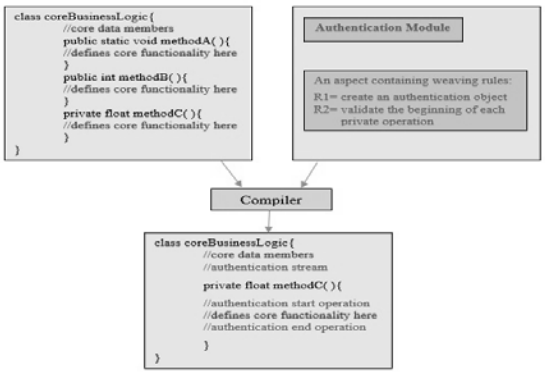

Fig.4 Code Demonstrating Authentication Concern

6. COMPARATIVE ANALYSIS

The OO programming manipulations are carried out on system level code. The OO software application is mainly based on the combination and collaboration of objects. Various contents are used in OO programming such as objects, classes, streams, and interfaces etc (Asagba, 2008). OO programming provides advantages like code reuse, flexibility, improved maintainability, modular architecture, decreased development time etc with the help of polymorphism, inheritance and encapsulation.

Moreover, OO programming code scatters system level code like logging and security etc with the production logic code. In the OO paradigm, a software module communicates directly to a block of executable code. A cross-cutting concern can be placed in multiple code blocks. This can enable modules into tangles mess of cross-cutting concerns. Object seems to be unhelpful in dealing with cross-cutting problems which are not detained to a software module or single class.

AO paradigm carries out its work at the composite software system as united implementation of multiple concerns like business logic, multithread safety, data persistence, error handling, logging, security and so on. The programming with AO splits the main executable code from that of overall system level code. The AO programming has joined points, point's cuts, advice and aspects at the same place (Papapetrou, 2004). The AO paradigm following the main programming tricks of OO paradigm as a base.

AO programming is modular as it facilitates reusability; it permits software developers to use aspect modules, wherever required in a software application exclusive of regenerating the same code. In AO programming, if we find multiple cases of code that need to be altered throughout the software application, it has always remained a challenge to search and modify each case in a large software application. AO paradigm facilitates software developers modify the aspect once and has its effects wherever it happens in software application. As a conclusion the AO paradigm helps to improve modularity of a software application.

7. CONCLUSION

Modularization is the main feature of today's software development methodologies, both selected paradigms supports this approach. The major difference between OO and AO Programming is that classes are used for implementing modularity in OO, while AOP implements with aspects. The focus of OO paradigm is to split the task in to objects which encapsulate attributes and services and in AO paradigm, program is divided in to cross-cutting concerns. Actually, cross-cutting concerns are supported by OO partially and fully by AO. After comparing the reported results of previous published research, it is concluded that AO paradigm supports modularization more easily and effectively than OO paradigm.

REFERENCES:

Amirat, A., M. Laskri, T. Khamaci, (2008). Modularization of Crosscutting Concerns in Requirements Engineering. The International Arab Journal of Information Technology, 5(2), 120-125.

Asagba, O., P. Ogheneovo. (2008). A Comparative Analysis of Structured and Object-Oriented Programming Methods. Journal of Applied Sciences and Environmental Management, 12(4), 41-46.

Ghezzi, C., M. Jazayeri, D. Mandrioli, (2003). Fundamentals of Software Engineering. Prentice Hall.

Kamble, G. (2009). AOP-Introduced Crosscutting Concerns. In the Proceedings of International Symposium on Computing, Communication and Control, 140-144.

Marin M., L. Moonen A. V. Deursen, (2007). An integrated crosscutting concern migration strategy and its application to JHOTDRAW. 7th IEEE International Working Conference on Source Code Analysis and Manipulation, Paris, France, 101-110.

Mesbah, A., A. V. Deursen, (2005). Crosscutting Concerns in J2EE Applications. In the Proceedings of the 7th IEEE International Symposium on Web Site Evolution, Wshington, DC, USA, 14-21.

Papapetrou, O., G. Papadopoulos, (2004). Aspect-Oriented Programming for a Component-Based Real Life Application: A Case Study. In the Proceedings of the Symposium on Applied Computing, ACM Press, 1554-1558.

Pressman, R. S. (2001). Software Engineering Practitioner's Approach. 5th Edition McGraw-Hill Series in Computer Science, 343-346.

Rumbaugh, J., M. Blaha, W. Premerlani, F. Eddy, W. Lorensen, (1999). Object-Oriented Modeling and Design. Prentice-Hall, 6th Edition, 64-67.

Subramaniam, H., H. Zulzalil, A. J. Marzanah, H. Saadah, (2014). Feasibility Study of Aspect Mining at Requirement Level. Indian Journal of Science and Technology, 7(5), 559-569.

Udayakumar, R., A. Kumaravel, K. Rangarajan, (2013). Introducing an Efficient Programming Paradigm for Object-Oriented Distributed Systems. Indian Journal of Science and Technology, 6(5S), 4596-4603.

Yang, S.. (2010). Research on Recovering Early Aspects in Aspect-Oriented Software Reserve Engineering. International Conference on Computer Application and System Modeling, Taiyuan, China, 202-206.