Improving the Software Quality using AOP

B Vasundhara
Associate Professor, Computer Science Department, AMS School of Informatics, Andhra Pradesh, India,
vasu_venki@yahoo.com

***Abstract* - The goal of software engineering is to solve a given application problem by implementing a software application system. Programming languages are important in software engineering. Ever since the advent of high-level programming languages, improvements have been motivated because of the need to build better software more rapidly. Concerns exist at every level of the system development process. The goal of modularization is to build application software that is maintainable and reusable. To implement such concerns we need to use a programming language that supports modularization. All the software engineering methodologies are expected to recognize the concerns of a system like the aspect-oriented software development (AOSD). AOSD additionally also classifies each of the concerns identified. Concerns in a system are of two types, core concerns and cross-cutting concerns. Core concerns make up the primary structure of the system. Cross-cutting concerns are those concerns that spread throughout the system. A major objective in software engineering is to increase code reuse in new systems as reuse saves development time. We can use the aspect-oriented programming (AOP) technique to improve software quality characteristics including, correctness, reliability, reusability, usability, efficiency, extendibility, timeliness, easy to use, etc.**

***Index Terms* - AOSD, AOP, Software Quality, AspectJ, Crosscutting Concerns, Join Point, a Pointcut**

I. EVOLUTION OF SOFTWARE ENGINEERING CONCEPTS – MODULARIZATION AND REUSE

The increasing intricacy, configuration and adaptability of real-time systems have become a strong motivation for applying new software engineering principles, like aspect-oriented development. AOSD improves the existing programming techniques by allowing the identification and description of concerns that crosscut many modules of the system. Applying AOSD in real-time application systems reduces the complexity of the system design and development. AOSD provides a way for a structured and efficient way to handle the crosscutting concerns in the system. We use AOP, a method for improving separation of concerns in software [1]. AOP is built on preceding technologies, like procedural and object-oriented programming which have already made substantial improvements in software modularity. The idea behind developing AOP is that, though the modularity mechanisms of object-oriented languages are extremely useful, they are essentially unable to modularize all concerns of interest in complex systems. To achieve this, AOP deals with crosscutting aspects of a system's behavior as isolated as possible.

Software modularity is a software design technique that builds the modules by breaking down the possible program functions [2]. Each module devised handles one of the many functions. Each module represents each of the separation of concerns, and thus improves the system maintenance by enforcing valid boundaries between the concerns. Aspects help in achieving increased modularity of the system and separation of concerns. Separating the functionalities as modules helps to control the system complexity. Software systems are conceptually complex by very nature, and increasing their complexity in the implementation means increasing the expense and the probability of failures. The code needed to integrate a complex implementation is expensive. The cost would be even higher if new features are to be added as and when required. Mostly adding new feature implies deep changes in several parts of the application implementation. So, we have to single out the modules that will implement the core business functions and that justify the design and implementation of software.

II. DRAWBACKS IN OBJECT-ORIENTED PROGRAMMING

An application may have some functionalities crossing it transversally, called as crosscutting concerns**.** A crosscutting concern is an independent entity that crosses other functionalities of software. Common crosscutting concerns include security of the system, logging across different functions when encountered, the transactions management, tracing, performance, synchronization, exception handling, etc. Such crosscutting concerns, if implemented only with object-oriented programming (OOP), results in a bad matching between the core concerns and the modules that implement the cross cutting concerns. So, when using OOP we are forced to deal with the execution of the crosscutting functionalities in separate modules, and further there may be a need to add other related modules or modify the existing ones. Therefore, it becomes necessary to modify the code in which these modules are used. This is undesired, but a necessary matching that the OO implementation unavoidably brings with it.

The crosscutting concerns in OOP gives rise to the problem of scattering code, due to the transversality of the crosscutting concerns that are implemented in classes. In such situations, AOP provides support to OOP for uncoupling modules that implement crosscutting concerns. AOP's purpose is the separation

DOI:10.32377/cvrjst0503

of concerns. In OOP, the fundamental unit is the class**,** while it is an aspect in AOP [1]**.** The aspect contains the implementation of a crosscutting concern. Code scattering appears when the application functionality is scattered due to its implementation in separate modules. Code tangling takes place when a module has to manage quite a lot of concerns at the same time, like logging into an application system. With an OO system, code tangling and code scattering can occur, thus causing the system to have duplicate code and result in functionalities not being clear.

AOP is a programming concept that is based on the identification and separation of both the core and crosscutting concerns of software. AOP is an extension of OO paradigm, in the sense that it provides new constructs for the modularization of crosscutting concerns. The main objective here is to define an implementation methodology using AOP to achieve better software with better quality [2]. With AOP, the crosscutting functionalities are extracted from the OOP implementations and applied as advices where they are actually executed. In OOP, we develop the code for every module where a functional component is encountered. While in AOP, an aspect code is developed and injected into the right locations of the base program using an aspect weaver. The main aim of an AOP language, like AspectJ, is to make sure that the aspect code and non-aspect code run together in a coordinated way using the process called aspect weaving.

III. ISSUES IN REAL-TIME APPLICATION SYSTEMS

The correctness of a real-time application system, depends both on the correct result produced by the computation and the time when the result is produced. Hence, enforcing timeliness is essential to the overall correctness the system. The increasing complexity in the design, adaptability and performance of a real-time system prompts us to use new software engineering programming methodologies, like AOP. Using AOP, the modules or concerns identified are incorporated into the program through interfaces.

Software reliability depends on system requirements, good design and implementation. A system fails if its behaviour is not consistent with its specification. The applications that need systems to maintain a predictable and correct functionality even in the presence of faults include online banking, mobile commerce, etc. For a system to be dependable, it must be available, reliable, safe, and secure. Software may undergo several upgrades during the system life cycle. These upgrades enhance the software reliability by re-designing or re-implementing the required modules. Further, new unexpected problems may arise. The two main concerns that have to be considered when designing the real-time application systems are the timeliness and criticality of the system. Additionally, we need to also consider that the systems are bounded by limited resources. To achieve the criticality and timeliness concerns with minimal resources, the real-time application systems use different techniques which involve a number of modules of the system. This makes it mandatory to use approaches like AOP.

IV. USING AOP FOR REAL-TIME APPLICATION SYSTEMS

The functional perspective of real-time application systems can be developed using conventional OOP, but the real-time perspectives like scheduling policies and synchronization mechanisms are better implemented using AOP. The purpose of AOP is to provide systematic means to modularize crosscutting concerns. AOP is an approach in this direction, which attempts to achieve the resuse of code and design in a much better way than OOP. Aspect-orientation is used in realtime application systems for distribution, timeliness and dependability domains [1].

We have a number of benefits in using AO techniques when compared to OO techniques. The primary benefit of this transition is the increased modularity. As concerns have been separated, the system's modules that were implemented to solve a given concern are not tangled with calls to modules that deal with unrelated, cross-cutting concerns. Also, these cross-cutting concerns are not scattered and are packaged into a single module. This, in turn, has several benefits throughout the software life-cycle, such as increased maintenance and reusability. An additional benefit of aspects is that of a reduction in the size of code. As aspects collect commonly repeated code into advice with a pointcut to where that code is relevant, the code-base will be smaller, when aspects are used, than when they are not used.

The key problems a system designer would face during the aspect-oriented design process include the identification and classification of concerns, testing the concern designs, reusing them, designing the concern modules, and refining the AO design. A standard aspect-oriented design process is an extensible, customizable and independent process which is easy to adopt. The existing literature describes less about the aspect oriented design processes in use; we can evaluate and validate the aspect oriented design process by applying it to case studies. Examples of the cross-cutting concerns include logging, exception handling, security.

The code to employ features like authentication, authorization, logging, exceptions, etc., is frequently scattered across the whole application. This will reduce the consistency, sustainability and quality of software. These characteristics are called crosscutting concerns. It is difficult to modularize the crosscutting concerns as they affect multiple functions and modules in the system. The AOP allows the localization and modularization of crosscutting concerns, thereby providing another level of abstraction called aspects.

While we are aware that crosscutting concerns can occur in various software systems, not much is known on how AOP and in particular AspectJ have been used.

DOI:10.32377/cvrjst0503

Several studies encompass the capabilities of AOP to improve the modularity, customization, and the evolution of software, but less is known about how AOP is used. Modularization of the crosscutting concerns of a software system will persist to be the source of initiative for progress in software engineering.

For a system concern like security, first we need to check for the users who attempt to access unauthorised data, and secondly prevent users from declassifying the data. If we use the OOP approach to implement the security concern, it will result in code scattering and code tangling, and its implementation will be weak. This weak implementation of security concern can be because of the intrinsic design of the system or a program error. The AOP approach using AspectJ language presents a strong implementation of security concern [1]. This reduces the load on the programmers to correctly recognize the positions in the base code where authorization is necessary. This can be achieved using AspectJ which is difficult to achieve using OOP language like Java. Even if the OOP concept provides a normal way to implement security concern, it does not really prevent any security flaws caused by bad program coding or poor system design.

AspectJ offers improved separation-of-concerns (SoC) in the system design phase, better encapsulation and also makes the implementation much cleaner than Java [5].

V. OVERVIEW OF AOP AND COMPARISON WITH OOP

A concern in software engineering means, a goal, functionality, or a requirement. The modularity mechanisms of OOP languages are useful, but they are essentially unable to modularize all the concerns in complex systems that involve more functionality. AOP attempts to achieve the reuse of code and design in a much improved way than OOP [3]. Therefore, AOP is more appropriate to implement the crosscutting concerns with better modularity. A software developer can use AOP language like AspectJ to isolate the code for implementing concerns like logging, security, etc., which otherwise are present at different locations in the base code. AOP achieves a more direct correspondence between design-level and implementation-level constructs, which leads to improved code quality which results in the reduction of the cost of designing, developing, and maintaining complex software systems.

AOP enhances the abstract degree and module character of software, which can improve the expansibility, reuse, easy understanding and maintenance of software and enhance other factors influencing the quality of software. The required methods codes are described within the aspect where the code executes instead of a class. Whenever the aspect code needs changes, there is only one place where we need to alter it. But, in OOP the software developer will have to trace all the classes employing the function code that needs changes. The aspect code is weaved or inserted wherever needed in the system at compile time or at run time. AOP modularizes the crosscutting concerns by encapsulating the replicated, scattered and tangled code into aspects.

The assimilation of base code and aspect code is called aspect weaving. The source code weaver merges the original source code with the aspect code. The aspects are interpreted and combined with the main program code and submitted to the compiler. After this, the compiler will generate the intermediate or machine language output.

VI. ASPECTJ AS AN AOP LANGUAGE

AspectJ is the most popular among the existing AOP languages [2]. AspectJ is an easy and convenient AOP language and is an extension to Java programming language with some new programming elements. It supports modularity and reuse of the aspects identified. Mostly crosscutting concerns implementation in AspectJ is dynamic [5]. To design a crosscutting behaviour, we have to identify the join points where we want to add or modify the behaviour. To apply such a design, we initially write an aspect for the module identified. Next, within the aspect we write the pointcuts to capture the required join points. Finally, we build an advice for each pointcut. Within the advice body we write the action that is to take place when the corresponding join points are reached. For instance, in the implementation of the logging concern affects every significant module in the system, the authorization concern affects every module with access control requirements, and the storage-management concern affects every stateful business object. We start by creating the aspect that encapsulates the logging concern. Next, we write the pointcut within the aspect that captures all join points where the operations are performed. Lastly, within the aspect we write an advice for the pointcut concerned, where we print the logging statement. Since the logging code lies inside the logging module and logging aspect; clients will no longer hold the code for logging. We find that the logging requirements are mapped directly into a single aspect. With such modularization, changes to the logging requirements will affect only the logging aspect. So, using AspectJ the core modules will no longer hold calls to the logging services.

There are crosscutting concerns that are not well captured by the traditional programming methodologies [5]. This motivates us to use AspectJ. Let's consider for example, the performance of a security policy in an application. We know that security concern cuts across an application. The security policy should be consistently applied to any improvements as the application advances, and also the security policy being applied may itself progress. Identifying such crosscutting concerns in a closely controlled way is complicated in a long-established programming language like Java. The advantage of implementing the crosscutting concerns in AspectJ over Java is that, the organization, evolution and implementation of the crosscutting concerns is easier and more stable.

VII. APPLICATION OF AOP TO REAL-TIME APPLICATION SYSTEMS

We are aware that the precision of a real-time application depends on the consistent result it produces and the instant when the results are produced. Therefore, enforcing timeliness is essential for the overall correctness of a real-time application system [4]. The aspects created represent the crosscutting concerns in the system. The aspects improve the system maintenance by enforcing logical boundaries between the concerns. Aspects are incorporated in the software through interfaces [4]. AOP approach is applied to case studies like Online Banking System and Shopping Catalogue. In particular, the concentration is on exception handling, logging, authentication, authorization, etc., as they contribute to the efficiency and reliability of these application systems. These Web-based systems need to use advanced technology to give the option of evading the time consuming and paper based features of conventional business. This results in managing the transactions more quickly and efficiently. Consumers' insight of security, accuracy, user-friendliness, and performance speed has become the essential factors for the success of such applications.

An online banking system is the technology which helps in avoiding prolonged and paper based characteristics of conventional banking, and thereby manages the business more efficiently. This application system allows us to connect to a bank through the Internet to view our accounts, credit, debit and transfer money, etc. Transaction security, accuracy, user friendliness and performance efficiency are the critical factors for the success of online banking. Quality attributes such as reliability, response time, security and availability are stringent system requirements for online banking. The Online Shopping Catalogue application provides Web access to various items. A user can browse a range of categories of items, select and add items to the catalogue and finally check out, do the payment and get the items. The requirements of this application can be considered as cross-cutting concerns.

VIII. RESULTS

AOP especially AspectJ can have a considerable affect on the program code size of an application by removing the code scattering and tangling [2]. AspectJ can be expected to reduce the program code volume by better code reuse and by reducing the code replication. There is a considerable decrease in the code redundancy. The 35-40% drop found in the code size is by separating the major functions of the system as aspects. There is lesser number of methods and also the program control flow is simplified [4]. While evaluating the AspectJ model for exception handling implementation, it is found that there is a decline in the number of lines of the concerned program. There is a considerable reduction of about 3% in the AOP version when compared to the 9% in the total code size in the OOP version. When crosscutting concerns are homogeneous, aspects considerably reduces the redundant code fragments.

It is found that there is an improvement in system modularity after the concerns are written as aspects [5]. This results in an increased consistency of functions in the system and a significant decrease in coupling between the crosscutting concerns. In the implementation of case studies we observed that the AO approach supports code mobility, usability and usefulness [3]. It is found that the AspectJ solution supports improved modularity because it reduces the overall coupling between the concerns. We know that for a software to be good, it should be flexible to take in necessary modifications through less effort. If the crosscutting concerns identified during software development are well modularized and implemented using AspectJ, we can accomplish the desirable software qualities like code stability, volatility, maintenance, etc.

The design stability is assessed in AspectJ implementations considered. The design stability is found mainly when the modifications were made in a particular crosscutting concern. These modifications are more simple to apply and less intrusive. The effect of overall AspectJ maintenance time mostly decreases. The study shows that AOP especially AspectJ, provides more support in the software evolution and maintenance than other solutions. Application development time using AspectJ is found to be less than other language implementation.

With new language constructs, AspectJ proposes modern ways to implement traditional programming mechanisms. For example, the case study implementation applies aspects to modularize the exception handling concern. It is found that AspectJ offers improved support for implementing exception handling. It is observed that when exception handling concern is non-uniform and complex, then use of AOP does not give the desired returns. The AOP implementation of exception handling concern reduces the code to define the exception interface and improves the separation between the base and aspect code. Effective join point representations have to be developed for more robust handling of the exception handling concern [4].

The advantages and restrictions of AOP, in particular AspectJ depend on the criterion like the software performance, application code size, system modularity, software evolution and language mechanism. After evaluation of the application of AspectJ in our case studies the behaviour of the above criteria is [5]:

- Performance: The results show that AspectJ generates positive outcomes with respect to the execution performance of the application systems because of better response time and minimum use of memory.
- Code size: AspectJ shows considerable decrease in the volume of application code, because of the

DOI:10.32377/cvrjst0503

separation of crosscutting concerns. According to the results, there is a noteworthy decrease in the overall application code volume by 40%.

- Modularity: Modularity is very prominent, especially in Separation of Concerns (SoC).
- Evolution: AspectJ has the capability to adapt to the incessant modifications in the user needs and functioning conditions. The outcome is positive in evolution context.

CONCLUSIONS

AOP has an optimistic impact on the software development process and improves the application software quality. The more the crosscutting concerns are isolated, the more effortless it is to carry out changes locally. Its effect on cognition and language mechanism is less likely to be positive. AspectJ can improve a system's performance where ever the crosscutting concern context is alike.

We observe improvements in the facets of AspectJ language evolution which includes volatility, extensibility, code stability, maintenance. AspectJ has the potential to develop evolving real-time application systems software whose maintenance is easy. AOP is a capable approach and a solution to the problems we face in conventional programming approaches. However, the solution presented by AOP necessarily may not come out well in terms of lower compilation time and less memory usage [4].

The basic concept and programming idea of AspectJ elaborates the software development approach based on AOP [3]. AOP has many works that need to be completed in future applications. The support languages need to be further enriched and their accuracies ensured, and more tools should be studied to support AOP and fulfil the demands in various stages from software design to maintenance. In view of the increasing software scale and complexity of software structure, the software development based on AOP technology would certainly play a more important function. AOP has remarkable prospective for constructing software for future applications. AspectJ compiler (ajc) needs to do more work than a pure Java compiler, so it is likely to take a little more time to compile an application. The small performance overhead caused is because of the need to analyze the classes, to see if any advice code needs to be woven into them.

REFERENCES

[1] D. Zhengyan, Aspect Oriented Programming Technology and the Strategy of Its Implementation, *The Proceedings of the International Conference on Intelligence Science and Information Engineering (ISIE)*, 2011, pp.457, 460, 20-21.

[2] 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.

[3] 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.

[4] Clark S and Baniassad E: Aspect Oriented Analysis and Design – The Theme Approach. *Addison-Wesley*, March, 2005.

[5] Elrad T, Filman R, and Bader A: Aspect Oriented Programming. *Communication of the ACM*, pp.29-32, October 2001/vol.44, No 10.

[6] Introduction to AspectJ. *http://eclipse.org/aspectj/doc/ released/progguide/starting-aspectj.html*

ABOUT AUTHOR

Ms Vasundhara is working as Associate Professor at AMS School of Informatics. Her areas of interest are Software Engineering, AOP, and Operating Systems. She has done her M Tech (CSE).

DOI:10.32377/cvrjst0503