ISSN : 2347 - 8446 (Online)
ISSN : 2347 - 9817 (Print)
Vol. 3, Issue 2 (Apr. - Jun. 2015)
International Journal of Advanced Research in Computer Science & Technology (IJARCST 2015)

Analysis of Crosscutting Concern Metrics of AOSD

[I]Swheta Gokulia, [II]Shrddha Sagar
[I,II]Dept. of CSE, Galgotias University, Gr. Noida,U.P, India

Abstract
In computing world , Aspect- Oriented Software Development (AOSD) is an emerging software development technology that seeks new modularizations of software systems. The modularization helps in isolating secondary or supporting functions from the main program's business logic. It allows multiple concerns to be expressed separately and automatically unified into working systems. This technology is implemented to identify, specify and represent the cross - cutting concerns and modularize them into separate functional units and their automated composition into a working system. In this paper Software Quality Models along with Software Metrics proposed for AOSD will be discussed.

Keywords
Aspect-oriented software development,quality models,aspect,advice,concern,join points,software metrics-crosscutting concerns.

I. Introduction

In early 1990's, Object-Oriented Approach (OOA) was introduced as a new paradigm in software development area with some important features like inheritance ,data abstraction and polymorphism [19]. Though, there are many advantages of OOA but programmer has to write the code line by line and every time it is required to write the code from scratch and it cannot be used in other software product. Also, the cost and effort required for developing the software is very high. To overcome these problems faced in OOA, AOSD was introduced. AOSD allowed software developers to develop the high quality and low cost software product.

AOSD now refers to a wide range of software development techniques that support the modularization of crosscutting concerns in a software system. It therefore covers the whole spectrum ranging from requirement engineering to business process management, implementation techniques, analysis and design, architecture, programming, testing and software maintenance techniques.

The use of AOSD techniques has increased over past decade in software systems[1].The main reason behind increasing use of AOSD is the fact that crosscutting concerns can be separated from other concerns in the system.

Three fundamental issues developers have to consider while implementing AOSD technique are as follows[2]:

- Developers need to determine whether AOSD techniques are suited to the problem at hand and the particular project context.
- Developers must ensure whether the potential benefits outweigh the costs of introducing a new technology and, if so, be able to convince management of its long-term profitability.
- Developers must avoid known pitfalls and deploy design strategies and tools to help counter their potential threat to product quality.

Today, real-world software needs evolution after a certain period of time to meet the current trends, technology, changes in user requirement and operational environment. During system evolution, software change is an essential operation. After the evolution of system, we must find out that how much new evolved system is extensible when the software functionalities are added or removed during maintenance, or when some modifications are done to the existing programs. Therefore, we require some measurement mechanisms for evaluating software quality. The development of high-quality software is a primary goal in software engineering. Such software are likely to be stable and maintainable. The quality metrics are applied during the development process by developers and managers to assess and improve software quality. For this purpose, software metrics are needed. Cohesion, coupling, and complexity are common types of such metrics[20].

The implementation of Software metrics are required in order to improve the productivity and quality of software, because they provide critical information about reliability and maintainability of the system[3].

II. Aspect –Oriented Software Developement

A concern is some part of the problem that we want to treat as a single conceptual unit [6]. Concerns are modularized throughout software development using different abstractions provided by languages, methods and tools.

Abstractions may not be sufficient for separating some special concerns found in most complex systems. These concerns are called as the crosscutting concerns since they naturally cut across the modularity of other concerns.

AOSD supports the modularization of crosscutting concerns by providing abstractions that make it possible to separate and compose them to produce the overall system[7].

Join points are points in the execution of the system, like method calls, where behavior supplied by aspects is combined. AOSD uses aspects as a new abstraction and provides a new mechanism for composing aspects and components (classes,methods, etc.) at specific join points. The join point model of an aspect-oriented language defines the types of join points that are supported by the aspect-oriented language and the possible interaction points between aspects and base modules. The dynamic interpretation of join points makes it possible to expose runtime information such as the caller or callee of a method from a join point to a matching pointcut[6].

An advice body is code that is executed when a join point is reached. The functional details of a concern are modularized by the advice. There are different ways of controlling the order in which advice bodies are contributed by aspects.

Inter-type declarations allow the programmer to modify a program's static structure, like class members and classes hierarchy. The insertion of new members and classes can be pushed down the class hierarchy.

A concern is encapsulated in a module called as an Aspect. Aspect-oriented approaches provide explicit support for localizing

ISSN : 2347 - 8446 (Online)
ISSN : 2347 - 9817 (Print)

concerns into separated modules known as aspects. An aspect is a module that encapsulates a concern. An aspect is composed of advice bodies, poincuts and inter-type declarations .
Aspect weaving is a composition mechanism that coordinates aspects with the other modules of the system. It is performed with the help of a specialized compiler, called an aspect weaver.
Without a proper way of separation and modularization ,crosscutting concerns are likely to be scattered and tangled up with other concerns. The natural consequences are ease of evolution , reduced comprehensibility, and reusability of software artifacts.

III. Literature Review

A. Quality Models

Azuma reffered to a Quality Model as "the set of characteristics and the relationships between them which provide the basis for specifying quality requirements and evaluating quality"[11]. Quality is an integration of multiple characteristics. Usually the quality is represented in the model which shows the quality characteristics and relationship among them. The models play an important role as they show what people think about quality. The software quality models are used to represent a more fixed and quantitative quality structure.
Jim McCall introduced first quality model in 1977 which differentiates between two levels of quality attributes known as quality factors. These are external attributes and can be measured directly. The second level of quality attributes known as quality criteria that can be measured subjectively or objectively [8,9].The second quality model introduced by Boehm who tries to overcome the problems of McCall's model . It presents a hierarchical structure for high level, intermediate level and primitive characteristics. Intermediate and primitive characteristics are similar to McCall's quality model which contributes the total quality of the system. But Boehm also includes Hardware performance i.e. missing in McCall's model. The Boehm model addresses the shortcomings of models that automatically and quantitatively evaluate the quality of software [10]. The intermediate level characteristics represent seven quality factors that represent the expected software quality by a system. The factors included in this model are: Portability, Maintainability, Usability, Human Engineering, Testability, Understandability and Flexibility.The Next model proposed by ISO introduce a new standard ISO 9126 in 1991 but fully adapted in 1992.This standard aims to define a quality model for software and a set of guidelines for measuring the characteristics. As it is improved version of ISO 9000 and it overcomes the problems of 1st release. Having a single universal model makes it easier to compare one product with another .The ISO 9126 quality model was suggested as an international standard for software quality measurement. Later a FURPS model is introduced by Robert Grady and extended by Rational Software. FURPS stands for Functionality, Usability, Reliability, portability and Supportability[11].
The latest model proposed by Kumar et al. is an extension of ISO 9126-1 software quality model in 2006. Four new sub characteristics are included in this model - modularity, code reducibility, complexity and reusability in addition to original characteristics and sub-characteristics of ISO 9126-1 model. AOSD is comparatively a modern Programming paradigm aimed to improve modularity.

B. Software Metrics

Software metrics have many applications in software engineering tasks such as program testing,reuse, understanding ,maintenance and project management. Almost all metrics proposed for AOSD are based on AspectJ implementation of aspect orientation. Summarization of well-known metrics of AOSD are as follows:
Coupling is an internal software feature which measures the level to which each program module relies on each one of the other modules. Coupling is considered to be a desirable goal in software construction, leading to better values for external attributes such as maintainability, reusability, and reliability [13]. Coupling is an manifestation of the strength of interdependence between the components in a system. Highly coupled systems have strong interconnections, with program units dependent on each other [14]. Coupling between components(CBC) and Depth of Inheritance Tree (DIT) are the coupling metrics of this suite. CBC is defined for a component (class or aspect) as a tally of the number of other components to which it is coupled. It enumerates the number of classes that are used in attribute declarations , the number of components declared in formal parameters, return types, throws declarations and local variables, and classes and aspects from which attribute and method selections are made. If a component A is coupled to a component B in an arbitrary number of modes, CBC enumerates only once.
DIT is defined as the maximal length from a node to the root of the tree. It enumerates how far down the inheritance ranking a class or aspect is declared.It is an extension of a Chidamber and Kemerer(CK) metric that considers the inheritance between aspects.
Separation of concerns refers to the ability to identify,encapsulate and manage those parts of software that are relevant to a particular concern . We defined the following metrics for Separation of concern:Coupling between Components (CBC), Concern Diffusion over Operations (CDO) and Concern Diffusion over Line Of Code (CDLOC). CBC is a design metric that enumerates the number of primary components whose main objective is to contribute to the implementation of a concern and the number of components that access the primary components by using them in attribute declarations, formal parameters, return types, throws declarations and local variables, or call their methods. CDO enumerates the number of primary operations whose main objective is to contribute to the implementation of a concern and the number of methods and advices that access any primary component by calling their methods or using them in formal parameters, return types, throws declarations and local variables. Constructors also are enumerated as operations. CDLOC enumerates the number of transmutation points for each concern through the lines of code. The purpose of this metric requires a shadowing process that segregates the code into shadowed areas and non-shadowed areas. The shadowed areas are lines of code that implement a given concern. Transmutations points are the points in the code where there is a transmutations from a non-shadowed area to a shadowed area and vice-versa.The intuition behind it is that they are points in the program text where there is a "concern switch." For each concern, the program text is evaluated line by line in order to count transmutations points. The higher the CDLOC, the more intermingled is the concern code within the application of the components; the lower the CDLOC, the more localized is the concern code[5,15].
Cohesion refers to the level of relatedness between members of a software component and mainly about how firmly the attributes and modules associate. It is a structural characteristic whose

significance is well recognized in software engineering community and is weighed to be a desired goal in software development, leading to better values for external characteristics. Certain metrics have been considered in order to assess cohesion of aspects -oriented software[13] .The advent for aspect cohesion measurement is based on following 3 factors:

Dependencies analysis

Gélinas and Badri introduced certain cohesion benchmark taking into account aspects' features and capturing various dependencies between their members. The proposed metric measures the level of relatedness of its modules called ACoh metric. A low value of ACoh symbolizes that the aspect members are poorly correlated[16].

Dependency graphs

Zhao and Xu's first considered the idea of implementing dependency model for aspect-oriented software that is a collection of dependency graphs. According to this idea , cohesion is defined as the level of relatedness between attributes and modules[13].

Lack of Cohesion in Operations

Sant Anna et al. considered an amplification of the prominent Lack of Cohesion in Methods (LCOM) metric developed by Chidamber and Kemerer for Object Oriented Programming.The proposed metric-Lack of Cohesion in Operations(LCOO) weighs the amount of method/advice pairs that do not access to the same instance variables. This metric measures the lack of cohesion of a component. A high LCOO value indicates disparateness in the functionality provided by the aspect[17].

The following are list of program structure metrics for AOSD . They represent the contribution of aspects to the overall structure of programs measured in line of code[18].

- Number of features (NOF), enumerates the number of features in a program.
- Number of Aspects (NOA), enumerates the number of aspects in a program.
- Number of classes and interfaces (NCI), enumerates the number of classes and inheritances in a program.
- Base Code Fraction (BCF), corresponds to the number of lines of the code that come from standard java classes and interfaces relative to the line of code in the program.
- Aspect code Fraction (ACF), corresponds to the numbers of line of code that come from aspects relative to the line of codes in a program.
- Introduction Fraction (IF), corresponds to the numbers of line of code that come from introductions or inter-type declarations relative to the line of codes in a program.
- Advice Fraction (AF), corresponds to the numbers of line of code that come from piece of advises relative to the line of codes in a program.

Despite of the fact that aspects improve modularization by crosscutting concerns little research has been done in characterizing and measuring crosscutting concerns .Crosscutting can be dynamic or static; static crosscuts affect the static structure of a program while dynamic crosscuts run additional code when certain events occur during program execution. Homogenous concern is one that applies a same piece of advice to several places; whereas a heterogeneous concern applies different pieces of advice to different places [18]. Adapting these concepts the following metrics are defined;

- Feature Crosscutting Degree (FCD),corresponds to the number of classes that are crosscut by all pieces of advice in a feature and those crosscut by the inter-type declarations.
- Advice Crosscutting Degree (ACD), corresponds to the number of classes that are crosscut exclusively by the pieces of advice in a feature.
- Homogeneity Quotient(HQ), as the division of the advice crosscutting degree (ACD) by the feature crosscutting degree (FCD):
- Program Homogeneity Quotient (PHQ), corresponds to the summation of the homogeneity quotients for all the features in a program, divided by NOF.
- Classes, interfaces, and aspects (CIA), determines the number of occurrences (NOO) of classes, interfaces, and aspects, as well as the Line Of Code(LOC) associated with each. It tells us if aspects (as opposed to classes and interfaces) are a small or a large fraction of the used modularization mechanisms in a software project, and if these implement a significant or only a small part of the code base of that project [21].
- Code Replication Reduction (CRR), determines the reduction in the LOC when using the homogenous advice, roughly the number effected join points, multiplied by the LOC associated with them .
- Degree of Scattering(DOS),measures the difference between the concentrations of concern over all components with respect to the worse case. A high DOS indicated the implementation of a concern is highly crosscutting .
- Degree of Focus (DOF), shows the variances of the dedication of a component to every concern with the respect of worse case. The average degree of focus gives an overall picture of how well concerns are separated in the program. Note that features refer to aspects, classes or interfaces.

IV. Conclusion

Authors have defined several software metrics for estimating software quality of software systems. After studying various research papers ,we have determined that from all of the software metrics crosscutting metrics are very important because study of Crosscutting Metrics allows the developer to perform a modularity analysis, identifying the crosscutting concerns in a system and also helps in quantifying the degree of crosscutting of each concern. We have analyzed that till now Crosscutting Metrics are precisely defined and validated..Therefore the future work will involve improving and validating proposed crosscutting metrics for AOSD.

References

[1] *R.E. Filman, "Aspect-Oriented Software Development Addison-Wesley", 2004.*
[2] *E. Baniassad , "Discovering Early Aspects," IEEE Software, Vol. 32, no. 1, pp. 61-69, 2006.*
[3] *SANJAY MISRA, IBRAHIM AKMAN and MURAT KOYUNCU," An inheritance complexity metric for object-oriented code:A cognitive approach", Vol. 36, Part 3, pp. 317–337, 2011.*
[4] *Shyam R. Chidamber and Chris F. Kemerer," A metrics suite for object oriented design." IEEE Trans. Software Eng., 20(6),pp. 476–493, 1994.*
[5] *Tarr, P. "N Degrees of Separation: Multi-Dimensional Separation of Concerns".Proceedings of the 21st International Conference on Software Engineering, 1999.*

ISSN : 2347 - 8446 (Online)
ISSN : 2347 - 9817 (Print)

[6] Kiczales, G. "Aspect-Oriented Programming". European Conference on Object-Oriented Programming (ECOOP), LNCS (1241), Springer-Verlag, Finland, 1997.

[7] Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J., Griswold, W. "Getting Started with AspectJ". Communication of the ACM, pp. 59-65,2001.

[8] Hoyer, R.W. and Hoyer, B.B.Y, "What is quality?" Quality Progress,Vol. 7, pp. 52-62, 2001

[9] Divya Prasad Narayani, Poonam Uniyal, "Comparative Analysis of Software Quality Models", International Journal of Computer Science and Management Research", Vol. 2, Issue 3, 2013.

[10] Deepshikha Jamwal, "Analysis of Quality Models for Organizations", International Journal of Latest Trends in Computing, Vol. 1, Issue 2, 2010.

[11] Suman, International Journal of Computer Science and Information Technologies (IJCSIT), Vol. 5 ,pp. 5634-5638, 2014.

[12] Ranbireshwar S. Jamwal, Deepshikha Jamwal & Devanand Padha, "Comparative Analysis of Different Software Quality Models", 3rd National Conference, 2009.

[13] Zhao, J., "Measuring Coupling in Aspect-Oriented Systems", Technical Report SE-142-6. Information Processing Society of Japan (IPSJ), 2003.

[14] Sommerville, I. "Software Engineering", 6.ed. Harlow, England, Addison-Wesley, 2001.

[15] Garcia, A. et al. "Agents and Objects: An Empirical Study on Software Engineering". Technical Report 06-03, Computer Science Department, PUC-Rio, 2003.

[16] Jean-François Gélinas, Mourad Badri, Linda Badri, "A Cohesion Measure for Aspects", Journal Of Object Technology, Vol. 5, No. 7, pp. 97-114, 2006.

[17] C. Sant'Anna, Alessandro Garcia, Christina Chavez, Carlos Lucena & Arndt von Staa, "On the Reuse and Maintenance of Aspect-Oriented Software: An Assessment Framework". XXIII Brazilian Symposium on Software Engineering, Manaus, Brazil, 2003.

[18] Lopez-Herrejon R. E. and Sven A., "Measuring and characterizing crosscutting in Aspect-Based programs:Basic Metrics and Case Studies", 2004.

[19] Szyperski.C., Component Software - Beyond Object-Oriented Programming, 2nd Edition, Addison-Wesley,1998.

[20] Jaspreet Kaur and Rupinder Kaur," Improving Applicability of Cohesion Metrics Including Inheritance", International Journal of Application or Innovation in Engineering & Management (IJAIEM), Vol. 2, Issue 3, 2013.