A STUDY OF COHESION METRICS FOR ASPECT-ORIENTED SYSTEMS

Kanika Arora¹, Abhishek Singhal², Avadhesh Kumar³

¹M.Tech. Student, ASET, Amity University, Noida-125, U.P, India, kanika.ga@gmail.com
²Assistant Professor, ASET, Amity University, Noida-125, U.P, India, asinghal1@amity.edu
³Professor & Head (CSE), Galgotia University, Greater Noida, U.P India, kumar.avadh@gmail.com

Abstract

Aspect-oriented is new programming approach to develop software. There are various existing approaches like modular and object-oriented but these approaches suffer from limitation in properly separating crosscutting concerns. Examples of crosscutting concerns are caching, tracing, logging, security, resource pooling, synchronization, exception handling etc. Code related to these concerns is generally duplicated with in several modules. This code must be properly encapsulated as well as must be scattered throughout the code. This is limitation in object-oriented and module-oriented system. Aspect-oriented approach provides efficient way for separating crosscutting concerns. Through Aspect-oriented system tangling of code will be reduced which makes it easier to understand core functionality. Also reusability and maintainence can be increased. An aspect contains some code and instructions about when to invoke the aspect, where to invoke and how to invoke. Crosscutting concerns are implemented as aspects. AspectJ, AspectC, AspectC++, AspectXML, CasearJ, Hyper/J, etc are various languages for aspect-oriented system. Different terms are introduced in respective aspect-oriented language. There are various metrics to measure quality characteristics of object-oriented system but very few metrics are for aspect-oriented system. Various quality characteristic are cohesion, coupling, complexity, size, reusability, changeability, maintainability, etc. Cohesion is one of the important quality characteristic. Cohesion is degree to which elements of module belong to each other. Through this paper we discussed about the aspect-oriented system and its advantage over module-oriented and object-oriented systems. In this paper we will focus on various cohesion metrics available for legacy system as well as for aspect-oriented system.

Index Terms: Object-oriented, module-oriented, Quality characteristic, metric, Cohesion.

1. INTRODUCTION

There are various approaches for software development. Module-oriented, object-oriented are few approaches of software development. Aspect-oriented is new approach that solves the problem that comes in module-oriented and object-oriented. Modular programming is based on the concept of procedural call. Procedure is executed as it is called during program’s execution. Object-oriented programming encourages the reusability and it has features like encapsulation, polymorphism, inheritance, etc. Practical experience shows handling or maintaining large code gets difficult as it becomes problem to clearly separate concerns. Example of concerns is exception handling, profiling (is determining where program spends its time), tracing (is determining which methods are called and when), security, synchronization issues, etc. To overcome this problem of handling code that can’t be properly put in modules and must be scattered throughout, Aspect is introduced. Aspect-oriented software development encapsulates concerns which lead to reduction in the amount of code to write [Alexander (2003)].

Aspect-oriented programming not completely replaces object-oriented programming but complements by supporting widespread implementation of crosscutting concern into a single unit called aspect. An aspect contains some code and instructions about when to invoke the aspect, where to invoke and how to invoke.

There are various aspect-oriented programming languages like AspectJ, AspectC, AspectC++, AspectXML, CasearJ, Hyper/J, etc. With different language different terms are introduced.

• AspectJ is extension of Java and is most widely used for aspect programming. AspectJ introduces the concept of Join points, Point-cuts, Advice and introduction [Kiczales et. al. (2001)]. Join points includes method call, access to class member, etc. Point-cuts are subset of join points i.e.
set of join points that are selected on the basis of some criteria like function name. Advice is code that executes around a join point or before or after a join point. Introduction is used to modify the static structure of program.

- CasearJ introduces the definition of virtual classes, aspectual polymorphism and binding. CasearJ [Aracic et al. (2006)] supports reusability and it combines aspect oriented constructs, point-cuts and advice with object-oriented techniques instead of having aspects as separate language abstraction.

- HyperJ [Pekilis (2002)] introduces Hyper-slice, hyper-space file, hyper-module file, a concern mapping file as new terms which act as three inputs. Hyper-space file describes Java classes that can be manipulated through Hyper/J and Hyper-module file describes how to integrate different dimension and which dimension of concern to integrate. Therefore, Hyper-Slice act as aspect and hyper-module as set of aspects with core modules. Concern mapping file tells which piece of Java source map to which dimension of concern.

In aspect-oriented software development cross cutting concerns are implemented as aspects. Aspect oriented software development has mechanism to weave aspect with core modules to make a working system [Aldrich (2004)]. Aspects are highly reusable as they are implemented separately. As reusability reduces the code hence reduces tangling, therefore, it becomes easier to understand the core functionality of modules. Aspect weaver is used to integrate the core module with aspects to form a final system. Aspects can not be compiled through traditional language compilers. Before compiler produces a binary executable file, some pre-processing is required. Aspect weaver, a special language processor is used for mapping aspects and other components. Aspects and components are abstracted from the problem domain, after which an aspect weaver merges these abstracted aspects and classes into an application that only contains traditional programming language constructs. Weaving is needed because the aspects are constructed with an aspect description language that cannot be processed by traditional compilers. By simply specifying which aspects should be woven into the code, one can easily build different systems without editing the components. This ability increases the reusability of components. With Aspect-oriented programming, the components can simply be reused without making any changes to the classes.

Java classes and aspect are weaved together and that intermediate code generated goes to Java compiler which then gives the byte code as its output. Through aspect weaver core modules are integrated with aspect modules and this intermediate code goes to java compiler to give byte code. [Fig-1]

AspectJ weaver is called ‘ajc’. Early versions of the AspectJ weaver only perform pre-processing and were depended on the standard Java compiler to generate code that runs on the Java platform, but new updated ajc is a complete compiler that adheres to the Java Virtual Machine specification. The ajc command compiles Aspects in aspectJ and Java language source files into class files.

![Fig-1. Aspect weaving process](image-url)

There are various models to evaluate the quality of object-oriented and module-oriented approaches Various Quality model for legacy system are given by Boehm et. al. (1976), McCall et. al.(1977), Droomey et. al. (1995), Ward et. al.(1999), ISO/IEC-9126 (2001), Chang et. al. (2008), etc. But very less work is done to evaluate the quality characteristics of aspect-oriented system. Various quality characteristic are cohesion, coupling, complexity, size, reusability, changeability, maintainability etc.

Cohesion is degree of relatedness among elements of the modules. High cohesion is desirable.

Coupling is degree of relatedness among the modules that is connection between modules. Low coupling is desirable.
**Complexity** is efforts needed to analyse code, efforts needed in modification and modification of modules.

**Maintainability** is modification of software product after delivery.

**Reusability** is using the module again to reduce the coding.

There are various software metrics for these characteristics for legacy systems. But very few metrics are discussed for aspect-oriented system. Software metrics act as indicator of quality of a system i.e. provide quantitative basis. Only few papers are addressing the concept of cohesion in existing literature. We will discuss various cohesion metrics for legacy and aspect-oriented system in the following sections. Cohesion improvement among modules or aspect shows high closeness and high inters relation. High cohesion tends to show high maintainability and reusability. In aspect-oriented system high cohesion occurs because of proper separation of cross cutting concerns.

2. COHESION METRIC FOR ASPECT ORIENTED SYSTEM

There are various cohesion metrics for aspect-oriented system but before discussing cohesion metrics for aspect-oriented system we take brief review of cohesion metrics for legacy system. Software metrics for legacy system are given by Eder, Chidmaber and Kemerer, Bela Vjhazi, Husein & Oxley, Hitz & Montazeri.

**Eder et. al. (1994)** extended the work concepts of cohesion for object-oriented systems that were initially developed for procedure oriented systems. According to him cohesion is defined as the binding of the elements with in the method & with in object class respectively. Eder, Kappel, Schreff in their Paper introduced the taxonomy of cohesion properties for object-oriented systems. Strong cohesion increases the possibility for reuse. On the basis of whether cohesion is between elements defined with in the same class or between the inherited or directly defined elements they are divided the cohesion in three categories:
- Method cohesion
- Class cohesion
- Inheritance cohesion

They extended the concept of cohesion of procedure oriented system that was based on single concept of module by considering interdependencies between methods & object classes in object method systems.

**Chidmaber and Kemerer** (1994) proposed a metric suite lack of cohesion in methods i.e. LCOM metric. LCOM is defined as number of disjoint sets formed by the intersection of the n sets. LCOM value gives the measure of the relative disparateness nature of methods in the class. A low value of LCOM high cohesion and vice versa when tie LCOM with instance variable & methods of a class then cohesion is measure of the attributes of an object class. Cohesiveness is desirable because it promote encapsulation. Low cohesion indicates that class can be split into two or more subclasses.

**Hitz and Montazeri** (1995) proposed cohesion measure based on chidamber & kemerer. Let X denotes a class, Ix – Set of instance variable and Mx – set of methods. E= {m accesses i \& n accesses i}

**Henderson-Sellers** (1996) gave properties for cohesion i. If measure yields 0, each method of the class references every attribute of class ii) If measure yields 1, every method of class references only a single attribute metrics for internal characteristics for aspect-oriented systems.

**Husein and Oxley** (2009) developed a tool CCMETRICS to calculate the cohesion of object-oriented system CCMETRICS allow evaluation of source code on package & class level.

**Bela Vjhazi et. al. (2010)** Proposed Metric conceptual lack of cohesion on methods (LCOM 5) which is based on LCOM5 Cohesion metric. The main advantage of this metric is that computation is simple as compared to other metrics.

There are lot of cohesion measurement approaches for object-oriented system as discussed above. But very few cohesion metric are there for AO systems, that too if available are mostly for AspectJ programming language.

Various metrics for aspect-oriented system are given by Sant Anna, Zhao and Xu, Gelinas, Kumar

**Sant Anna et. al. (2003)** proposed new metric Lack of cohesion in operations i.e. LCOO which is extension of Lack of cohesion methods i.e. LCOM given by Chidmaber and Kemerer for object-oriented system. LCOO take into account relatedness among class and aspects. LCOO measures amount of method/advice pairs that do not access the to the same instance variable. Lack of Concern based cohesion for measuring cohesion is metric suite proposed by Sant’ Anna. This metric suite is for concern driven architecture, therefore, counts the number of concerns addressed by the assessed component. How to evaluate cohesion empirically is not specified.
Zhao and Xu (2004) defined Cohesion on the basis of relatedness among attributes and modules. Cohesion is measured on the basis of dependency graphs. They considered measurement on the basis of inter-modules and module attribute dependencies. The approach suggested by them is complicated and can be problematic to use for real world problems.

Gelinas et al. (2006) proposed ACOH metric for measuring cohesion for aspect-oriented system. They defined cohesion on the basis of dependencies. They considered two aspect cohesion criteria for measurement – Module-Module and Data connection criteria and module-module connection criteria. They defined ACOH metric on the basis of total number of module pairs in an aspect and maximum number of connections between aspect modules.

Kumar et al. (2008a, 2010b) derived new metric from ACOH metric given by Gelinas. Kumar proposed UACOH metric that is applicable to most of aspect programming languages. They identified various different connections for cohesion and used those connections as criteria of cohesion measurement in ACOH. They derived cohesion metric from Gelinas et al. (2006) ACOH takes into account actual number of connection and maximum number of connection possible. They extended there work to correlate cohesion and changeability and came to conclusion that metric to measure cohesion can not be used for assessing changeability of AOS.

<table>
<thead>
<tr>
<th>Authors</th>
<th>Year</th>
<th>Metrics</th>
<th>Description</th>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sant Anna et. al.</td>
<td>2003</td>
<td>LCOO</td>
<td>Take into account relatedness among class and aspects</td>
<td>Counts the number of concerns addressed by the assessed component</td>
<td>How to evaluate cohesion empirically is not specified</td>
</tr>
<tr>
<td>Zhao and Xu et. al.</td>
<td>2004</td>
<td>-</td>
<td>Defined Cohesion on the basis of relatedness among attributes and modules</td>
<td>Measurement on the basis of inter-modules and module attribute dependencies</td>
<td>Approach is complicated and can be problematic to use for real world problems.</td>
</tr>
<tr>
<td>Gelinas et. al.</td>
<td>2006</td>
<td>ACOH</td>
<td>Defined cohesion on the basis of dependencies criteria</td>
<td>Is well adapted to reflect design problems i.e. it take into account aspect’s characteristics</td>
<td>Not taken into account indirect relationship between aspect members.</td>
</tr>
<tr>
<td>Kumar et. al.</td>
<td>2008</td>
<td>UACOH</td>
<td>Identified various different connections for cohesion and used those connections as criteria of cohesion measurement</td>
<td>Applicable to aspect programming languages that are either extension of Java or have features of Java.</td>
<td>Work is to be done for other AOP languages like AspectC, AspectL, etc</td>
</tr>
</tbody>
</table>
3. CONCLUSION

In this paper, features of aspect-oriented system have been discussed and its advantages over module-oriented and object-oriented systems have also been explored. Software quality is very important concern, some important internal and external quality factors for software system has been highlighted. Cohesion which is one of the important internal quality characteristic has been reviewed for aspect-oriented system. Comparative analysis of various cohesion metrics proposed by various researchers and practitioners specific to aspect oriented system has been done. Using internal characteristics such as cohesion can be used to predict indirectly measurable external quality characteristics such as maintainability, reusability, modularity and usability of aspect-oriented system.

REFERENCES


**BIOGRAPHIES**

**Kanika Arora** has received her B.Tech degree in 2010. Currently she is currently pursuing her M.Tech from Amity University. Her area of interest includes aspect systems and adaptive websites.

**Abhishek Singhal** is working as an assistant professor in Amity University Noida. He has obtained B.E. M.Tech. from reputed universities. Currently he is pursuing PhD from Amity University Noida. His research interest includes software testing, aspect-oriented systems, image processing and software engineering.

**Dr. Avadhesh Kumar** is presently working as Professor & Head (CSE) in Galgotia University, Greater Noida, He obtained his B.Tech. in Computer Science & Engineering from H.B.T.I. Kanpur, U.P., India, M.Tech. in IT from Punjabi University, Patiala, Punjab, India and Ph.D. thesis at Thapar University, Patiala, Punjab, India. His research interest includes software testing, aspect-oriented systems, autonomic computing and software engineering.