Current States of Aspect Oriented Programming Metrics

Esubalew Alemneh

School of Computing and Electrical Engineering, Bahir Dar University
P. O. Box 26, Bahir Dar, Ethiopia

Abstract: ***Aspect Oriented Programming (AOP) is a new technology for separating crosscutting concerns that are usually hard to do in object-oriented programming. As AOP has better capability to handle crosscutting concerns than object-orientation it helps to write more modularized and more maintainable code. And numerous publications discuss about the advantages of AOP design and implementation. However, with respect to metrics for this new programming paradigm the work is in its infancy. In this paper we have surveyed, summarized and reviewed all available internal metrics for aspects-oriented systems.***

Keywords: Aspect Oriented programming, software metrics, crosscutting

1. Introduction

Programming in its outset was done by plugging and unplugging cables. Then it is evolved to assembly language where programming is done by using zero and one. In this era only experts can write a program using such method. Meanwhile human like non-structural programming language called FORTRAN is emerged. This programming language has replaced cumbersome and challenging programming by much easier and simpler on. People were not still satisfied with the existing programming paradigm and come up with structured imperative languages like ALGOL, Pascal and then to the object-oriented paradigm, with languages like Smalltalk, C++ and Java which is the fourth level of programming structure and evolution. The evolution of program paradigm has not give up yet. An ever-growing complexity of software has revealed the weaknesses of Object oriented programming. To deal with this problem a new programming paradigm, aspect-oriented programming has emerged.

In an Object-Oriented (OO) application, classes collaborate to achieve the application's overall goal. However, there are parts of a system that cannot be viewed as being the responsibility of only one class, they cross-cut the complete system and affect parts of many classes. Aspect-oriented software development (AOSD) is a technique to support separation of concerns in software development [2]. AOSP which may arise at any stage of software life cycle, including requirements specification, design implementation etc, involves modularizing crosscutting aspects of a system. Some of examples of crosscutting aspect are logging, exception handling, synchronization, resource sharing, performance optimization and security.

Aspect-oriented programming (AOP) is a promising new software development technique claimed to improve code modularization and therefore reduce complexity of object-oriented programs [1, 11]. In this paradigm a new kind of component called aspect to model the crosscutting concerns in software system is introduced. An aspect is a modular unit of crosscutting implementation. It is defined very much like a class (aspect oriented system also contains classes), and can have methods, fields, and initializers and pointcuts, advice and introductions for the crosscutting implementation. This concepts and methods can be implemented in one of AOP languages. The most popular example is Aspect J, which has been created at Xerox PARC. It is a seamless aspect-oriented extension to java. AOP and hence AspectJ highly relies on object-oriented programming principles.

Although plenty of research in a software metrics has been focused on procedural or object-oriented software as well as software architectures until now, a little is done on metrics of aspect oriented programming [5]. The emphasis is on problem analysis, software design, and implementation techniques. But, there must be a rigorous and quantitative way, at least, to evaluate these design techniques. Software developed using AOSD is deemed to have high quality. But, unfortunately there are no sufficient methods to prove this quantitatively, albeit the statement seems to be correct.

AOP is inherently originated from Object-Oriented Programming (OOP) and procedural programming concepts. So, though it cannot be applied directly from its origin, existing metrics from OOP and procedural programming can be used as a starting point to measure AOP metrics. Accordingly, some metrics has been purported for AOP though validated rarely. Examples include coupling, cohesion, complexity, crosscutting metrics, and program structure metrics. For some of these metrics theoretical validation has been addressed. To the best of our knowledge no empirical validation has been addressed to AOP metrics mentioned above. Validating these metrics is very important to have meaningful measures and to predicate external attributes like usability, maintainability and reliability.

The rest of the paper is organized as follows. Section 2 summarizes the works done on metrics of AOP. Some selected example metrics and frameworks are discussed in section 3. The summary of metrics of AOP is also presented in tables in section 3. Finally, at section 4 conclusion and future works are given.

2. Related Works

Software metrics have always relied strongly on the paradigm used in the respective period. Among the earliest metric, McCabe Cyclomatic complexity number was design for measuring the testing efforts written using the non-structural FORTRAN program [12]. Implementations of object-oriented programming methods have large procedural components. In turn implementation of AOP is highly relied

on OOP principles [1]. Thus most of the metrics for AOP are derived from the concepts of OOP. Figure 1 shows the framework of OOP measure as formulated in [13]. This framework can be generalized or can be customized to measure metrics of AOP.

Based on dependence model some metrics are proposed to quantify information flow of aspect-oriented programs [5]. The dependence graphs of the model are defined at three levels (module-level, aspect-level and system-level) and for each level different kinds of metrics are defined.

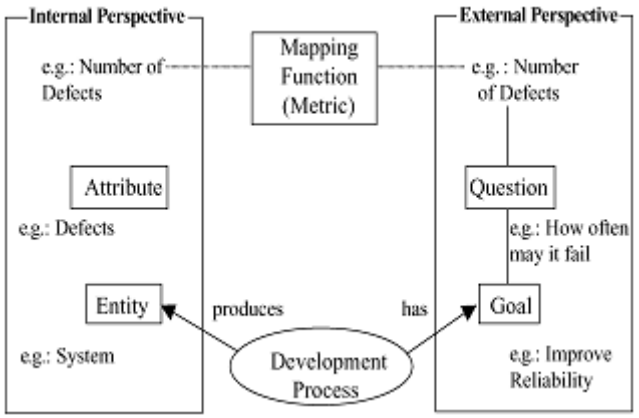

Figure 1: An object oriented measurement framework

Coupling for aspect oriented is derived from Chidamber and Kemerer's framework for Object-Oriented measures. Zhao, Bartsch and Harrison define coupling measures for aspect-oriented programming [2, 7]. Zhao designs coupling framework for AO system and formally define various coupling measures in terms of different types of dependencies between aspects and classes only, while Bartsch and Harrison focuses on the evaluation of five aspect-oriented coupling measures in the intension of increasing quality of software.

The first attempt to measure cohesion of aspect-oriented system is made by Zhao [4]. In this work an approach for assessing the aspect cohesion based on dependence analysis is proposed. Another work regarding cohesion is by Gelinas J., Badri M., and Badri L, in [9] in which they have proposed a new approach for aspect cohesion measurement based on dependencies analysis. Several cohesion criteria taking into account aspects' features and capturing various dependencies between their members are introduced. They also proposed new aspect cohesion metric and compare it, using several case studies.

There is no appropriate metric tool to present quantitative results on the structural complexity of AOP programs [1]. Lack of multiparadigm metrics that are valid on both Object-Oriented and generic paradigm is one of the reasons. Pataki, Sipos and Porkolab have analyzed the GoF design patterns and their implementations in pure Java and an AOP version in AspectJ in order to answer questions such as why is it easy to understand some solutions than the others if they are implemented using different paradigms. Their findings show that aspect-orientation does not necessarily reduce the complexity in its own.

A basic code metrics that categorize crosscutting according to the number of classes crosscut and the language constructs used are presented by Roberto and Sven [3]. The metrics are applied to four non-trivial open-source programs implemented in AspectJ and it is found that the number of classes crosscut by advice per crosscutting is small in relation to the number of classes in the program. Apel Sven, Batory Don, Rosemuller Marko, in [17] concentrated in the crosscutting concern in presenting a framework for classifying the structural properties of crosscutting concern into those benefit from AOP and that should be implemented by OOP mechanism.

No metrics for AOP is validated though some of the metrics may be well -defined. A short overview on the necessary steps for validating definitions and applications of metrics that are to be used in an evaluation process are depicted in [6]. Unless the metrics are well-defined and validated the usefulness of the metrics is in question.

3. Metrics of AOP

Software metrics have many applications in software engineering tasks such as program understanding, testing, reuse, maintenance, and project management. Though not much, some metrics are proposed to AOP. Almost all metrics proposed for AOP are based on AspectJ implementation of aspect-orientation. In the following subsections we will summarize the well-known internal metrics of AOP.

3.1 Coupling

Coupling is an internal software attribute which measures the degree to which each program module relies on each one of the other modules. Coupling is thought to be a desirable goal in software construction, leading to better values for external attributes such as maintainability, reusability, and reliability [2]. Low coupling is the characteristics of good software design. Coupling in AO system mainly appears between aspects and/or classes (basic components of AOP) through advice, inter-type declaration, pointcut, and method call.

Although coupling has been widely studied for object-oriented system, only few researches has studied coupling in aspect oriented systems. Zhao J has proposed a measure suite for assessing the coupling in AO systems [2]. In this work first they have presented a coupling framework for AO system which specially designed to count dependencies between aspects and classes in the system (though aspect might contain interface they haven't considered it). And then various coupling measures in terms of different types of dependences between aspect and class are formally defined based on the metrics. Finally the mathematical properties of this measures have been discussed, in which they showed that the measure satisfy the properties that a good coupling measure should have.

In other work evaluation of coupling measures for AspectJ has been conducted. The focuses were on the evaluation of five aspect oriented coupling measures with the aim to constructively increase the quality of software evolution [7]. These five coupling measures, which are suggested by

cecceto and Tonella previously, are coupling on advice execution (CAE), coupling on intercepted module (CIM), coupling on method call (CMC), coupling on field access (CFA), and crosscutting degree of an aspect (CDA). Some of these metrics are well-defined but no metrics is validated from measurement point of view. So, all measures including coupling measures need to be validated to gain confidence in the results taken from the measurement.

3.2 Complexity

Metrics for assessing the complexity of aspect-oriented software, which are specifically designed to quantify the information flows in aspect oriented program, is proposed by Zhao in [5]. The metrics are defined based on a dependence model of aspect oriented software. The model has three level dependence graph; module level dependence graph, aspect level dependence graph and system level dependence graph. For each level different metrics has been identified. Some object oriented or procedural programming complexity metrics like McCabe's cyclometic complexity can also be directly adapted to measure section of aspect oriented program.

A multiparadigm metric to measure the complexity of aspect oriented programs is studied in [1]. The metrics are used to compute structural complexity of all the object oriented, aspect oriented and procedural component of AOP code, hence called multiparadigm. Pure object oriented programming by java and aspect oriented programming by AspectJ are used to implement Gang-of-Four (GoF) design patterns which are functionally equal. The metrics used is AV complexity. The implementations are tested and the metrics revealed that aspect orientation doesn't necessarily the complexity in its own. The complexity highly depends on the nature of the actual problem.

3.3 Cohesion

Cohesion refers to the degree of relatedness between members of a software component and mainly about how tightly the attributes and modules cohere. It is a structural attributes whose importance is well recognized in software engineering community and is considered to be a desired goal in software development, leading to better values for external attributes. Several metrics have been proposed in order to assess cohesion of aspects -oriented software [4]. Their approach for aspect cohesion measurement based on:

a. Dependencies analysis
   Jean-François Gélinas, Mourad Badri and Linda Badri in [9] has introduced several cohesion criteria taking into account aspects' features and capturing various dependencies between their members. The proposed metric measures the degree of relatedness of its modules called ACoh metric. A low value of ACoh indicates that the aspect members are poorly related.

b. Dependency graphs
   Zhao and Xu's approach [4] is the first proposal in the field of aspect cohesion measurement. It is based on a dependency model for aspect-oriented software that consists of a group of dependency graphs. According to Zhao and Xu's approach, cohesion is defined as the degree of relatedness between attributes and modules. Zhao and Xu present, in fact, two ways for measuring aspect cohesion based on inter-attributes, inter-modules, and module-attribute dependencies.

c. Lack of Cohesion in Operations
   Sant Anna et al. proposed in [8] an extension of the well-known Lack of Cohesion in Methods (LCOM) metric developed by Chidamber and Kemerer [10] for OOP. The proposed metric LCOO (Lack of Cohesion in Operations) measures 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.

3.4 Crosscutting Metrics

Despite of the fact that aspects improve modularization by crosscutting concerns little research has been done in characterizing and measuring crosscutting concerns [3]. 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 [3]. Adapting these concepts the following metrics are defined;

- FCD, Feature Crosscutting Degree, 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.
- ACD, Advice Crosscutting Degree. Corresponds to the number of classes that are crosscut exclusively by the pieces of advice in a feature.
- HQ, Homogeneity Quotient as the division of the advice crosscutting degree (ACD) by the feature crosscutting degree (FCD):
- PHQ, Program Homogeneity Quotient. It corresponds to the summation of the homogeneity quotients for all the features in a program, divided by the number of features (NOF).
- Classes, interfaces, and aspects (CIA) project. The CIA metric determines the number of occurrences (NOO) of classes, interfaces, and aspects, as well as the 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 [17].
- CRR, Code Replication Reduction. Determine the reduction in the LOC when using the homogenous advice, roughly the number effected join points, multiplied by the LOC associated with them [17]. Overall code reduction born from the sum of the saved LOC of all advice and the intertype declarations. They argued that the structure of a concern decides over how it is implemented.
- Degree of Scattering (DOS). Measure 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 [18].

- Degree of Focus [DOF]. Show 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 [18].

Note that features refer to aspects, classes or interfaces.

3.5. Program Structure Metrics

The following are list of program structure metrics for AOP as mention in [3]. They represent the contribution of aspects to the overall structure of programs measured in line of code.

- Number of features, NOF, Counts the number of features in a program.
- Number of Aspects, NOA, Counts the number of aspects in a program.
- Number of classes and interfaces, NCI, Counts 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.

To sum up, though metrics which are not validated can be useful in some circumstances all metrics should be validated in order to have assurance on the results of measures. No AOP metrics are validated thought some are defined. For defined it means the metrics can be mapped without any ambiguity to the framework of the attribute and validation refers to the conformance to validation framework. Except coupling, to the best of our knowledge, all other metrics of AOP have no frameworks of well-definedness and validation. So, we cannot attribute these metrics as defined or validated. In addition different measures of an attribute which are proposed by different people can be somehow overlapping, but as long as they are not exact copy of one another we presented all in the following tables

Table 1: Metrics of AOP

Attribute	*Measures*	*Proposed by*
Cohesion	Aspect Cohesion (ACoh)	Gélinas, Mourad, Linda (2006)
	Lack of Cohesion in Operations (LCOO)	Anna,Garcia,Chavez, Lucena, Staa (2003)
	Inter-Attribute cohesion*	Jianjun Zhao & Baowen Xu - 2004
	Module-attribute	
	Inter-module cohesion*	
Complexity	Module-level Metrics*	Jianjun Zhao (2002)
	Aspect-level metrics*	
	System-level Metrics*	
	AV complexity	Pataki, Sipos, Porkolab (2004)
Cross cutting	Feature Crosscutting Degree (FCD)	Lopez-Herrejon, Apel (2004)
	Advice Crosscutting Degree (ACD)	
	Homogeneity Quotient (HQ)	
	Program Homogeneity Quotient (PHQ)	
Program Structure Metrics	Number of features	Lopez-Herrejon, Apel (2004)
	Number of Aspects	
	Advice Fraction (AF)	
	Base Code Fraction(BCF)	
	Aspect code Fraction	
	Introduction Fraction (IF)	
	Number of classes and interfaces(NCI)	

Items with * indicate it may contains more than one measure.

Table 2: Coupling and its measures

Attribute	*Measure*	*Proposed by*	*Defined*	*Validated*
Coupling	coupling on advice execution (CAE)	Cecceto and Tonella, 2004	No	No
	coupling on intercepted module (CIM)		Yes	No
	coupling on method call (CMC)		No	No
	coupling on field access (CFA)		No	No
	Crosscutting degree of an aspect (CDA).		No	No
	Attribute-class dependence measure	Jianjun Zhao, 2003	No	No
	Module-class dependence measure			
	Module-method dependence measure			
	Aspect-Inheritance dependence measure			

4. Conclusion and Future Work

In an OO application classes collaborate to achieve the application's overall goal. However, there are parts of a system that cannot be viewed as being the responsibility of only one class, they cross-cut the complete system and affect parts of many classes [16]. AOP is the solution for this problem. But measure of this new programming language is rarely done. In this paper we have summarized the metrics available for AOP. Almost all metrics are not well-defined and validated although the metrics proposed are few in

number. Some future research directions on AOP metrics are definition of new metrics for the language and design and evaluation of frameworks to evaluate the well-definedness and validation of existing and new metrics.

Reference

[1] Pataki N. Sipos A., and Porkolab Z., "Measuring the complexity of Aspect-Oriented programs with Multiparadigm Metric", 2004.
[2] Zhao, J., "Measuring Coupling in Aspect-Oriented Systems", Technical Report SE-142-6. Information Processing Society of Japan (IPSJ), June (2003).
[3] Lopez-Herrejon R. E. and Sven A., "Measuring and characterizing crosscutting in Aspect-Based programs: Basic Metrics and Case Studies", 2004.
[4] Zhao, J. and Xu, B., "Measuring aspect cohesion". Proceedings of the International Conference on Fundamental Approaches to Software Engineering (FASE), LNCS 2984, Barcelona, Spain, March 2004. Springer Verlag, pp.54-68.
[5] Zhao, J. "Towards a Metrics Suite for Aspect-Oriented Software". Technical-Report SE-136-25, Information Processing Society of Japan (IPSJ), March 2002.
[6] Mehner K., "On using Metrics in the Evaluation of Aspect-Oriented Programs and Designs", 2005.
[7] Bartsch, M., Harrison, R, "An Evaluation of Coupling Measures for AspectJ", LATE Workshop AOSD (2006).
[8] 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, October 2003.
[9] Jean-François Gélinas, Mourad Badri, Linda Badri, "A Cohesion Measure for Aspects", Journal Of Object Technology, Vol. 5, No. 7, pp 97-114, September-October 2006.
[10] S.R. Chidamber and C.F. Kemerer,"A Metrics suite for object Oriented Design", IEEE Transactions on Software Engineering, Vol. 20, No. 6, pp. 476-493, June 1994.
[11] The AspectJ Team, "The AspectJ Programming Guide," 2001.
[12] McCabe, T.J.," A Complexity Measure, IEEE Trans. Software Engineering", SE-2(4), pp. 308-320, 1976.
[13] Sandeep Purao and Vijay Vaishnavi; "Product Metrics for Object-Oriented Systems", SANDEEP PURAO, ACM Computing Surveys, Vol. 35, No. 2, June 2003, pp. 191–221.
[14] Chidamber S.R., Kemerer, C.F., "A metrics suit for object oriented design", IEEE, Trans. Software Engeneering, vol.20, pp.476-498, (1994).
[15] Apel Sven, Batory Don, Rosenmuller Marko. "On the Structure of Crosscutting Concerns:Using Aspects or Collaborations?", 2007
[16] Marc Eaddy, Alfred Aho, "Towards Assessing the Impact of Crosscutting Concerns on Modularity", 2007

Author Profile

Esubalew Alemneh Jalew has received his Bachelor of Science degree in Computer Science from Addis Ababa University, Ethiopia and his masters in computer science from Universiti Putra Malaysia. Now he is a lecturer in Bahir Dar University, Ethiopia.