Volume 1, No. 3, Sept-Oct 2010
ISSN No. 0976-5697
International Journal of Advanced Research in Computer Science
REVIEW ARTICLE
Available Online at www.ijarcs.info

Coupling Metrics for Aspect Oriented Programming -A Systematic Review

Kotrappa Sirbi*
Department of Computer Science & Engineering
K L E's College of Engineering & Technology
Belgaum, India
kotrappa06@gmail.com

Prakash Jayanth Kulkarni
Department of Computer Science & Engineering
Walchand College of Engineering
Sangli, India
pjk_walchand@rediffmail.com

Abstract: Metrics are an important technique in quantifying desirable software and software development characteristics of aspect- oriented software development (AOSD). Coupling is an internal software attribute that can be used to indicate the degree of system interdependence among the components of software. Coupling is thought to be a desirable goal in software construction, leading to better values for maintainability, reusability and reliability. Although several coupling frameworks and coupling metrics have been proposed for aspect-oriented software, the tool support and empirical evaluation of these metrics are still being missed. However, there have been very few attempts to systematically review and report the available evidence in the literature to support the claims made in favor or against AOP coupling metrics for maintainability and reuse, modularity etc., compared with OOP approaches. In this paper, we present a systematic review (extended version) of recent coupling metrics for AO designs. In this review work consolidates data from recent research results, highlights circumstances when the applied metrics suitable to AO designs, draws attention to deficiencies where AO metrics need to be improved.

Keywords: Aspect Oriented Programming (AOP), Software Quality Metrics, Aspect Oriented (AO) Metrics, Aspect J.

I. INTRODUCTION

Now days, our society is becoming dependent on software that's why demand of quality software is increasing day by day. In the literature of software quality models, many researchers and practitioners have proposed their quality models, which are intended to evaluate external software qualities such as maintainability, usability, efficiency, functionality, reliability, portability and reusability. These external software quality characteristics could be measured with the help of software metrics. Metrics are designed on the basis of design structure of programming languages such as module-oriented programming (MOP), object-oriented programming (OOP) and aspect-oriented programming (AOP). Design of metrics depends on internal quality characteristics such as encapsulation, cohesion, coupling and complexity. In turn, researchers and practitioners have proposed a large number of new metrics and assessment frameworks for quality design principles such as complexity. High complexity of any software system is an indication of low quality.AOP languages aim to improve the ability of designers to modularize concerns that cannot be modularized using traditional module-oriented (MO) or object-oriented (OO) paradigms. Such concerns are scattered in multiple modules (classes) and are known as crosscutting concerns. Examples of crosscutting concerns include logging, tracing, caching, resource pooling etc. The ability to modularize such concerns is expected to improve comprehensibility, parallel development, reuse and ease of change, reducing development costs, increasing dependability and adaptability. Since AO is a new abstraction, the definition of complexity is required to redefine in the context of AOP.

Aspect-oriented programming (AOP) [2] is now well established in both academic and industrial circles, and is increasingly being adopted by designers of mainstream implementation frameworks (e.g. JBoss and Spring). AOP aims at improving the modularity and maintainability of crosscutting concerns (e.g. security, exception handling, caching) in complex software systems. It does so by allowing programmers to factor out these concerns into well-modularised entities (e.g. aspects and advices) that are then woven into the rest of the system using a range of composition mechanisms, from pointcuts and advices, to intertype declarations [2,31] and aspect collaboration interfaces.

Unfortunately, and in spite of AOP's claims to modularity, it is widely acknowledged that AOP mechanisms introduce new intricate forms of coupling [21, 28], which in turn might jeopardise maintainability [1, 3]. To explore this, a growing number of exploratory studies have recently investigated how maintainability might be impacted by the new forms of coupling introduced by AOP mechanisms [14, 15, 16].

The metrics used by these studies are typically taken from the literature [10, 11, 22, 23, 26, 32] and are assumed to effectively capture coupling phenomenon in AOP software. However, the use of AO metrics is fraught with dangers, which as far as AOP maintainability is concerned have not yet been thoroughly investigated. In order to measure coupling effectively a metrics suite should fulfill a number of key requirements. For instance, the suite should take into account all the composition mechanisms offered by the targeted paradigm[19,20,21], the metrics definitions should be formalised according to well-accepted validation frameworks, e.g. Kitchenham's validation framework[17,18,19], and they should take into account important coupling dimensions, such as coupling type or strength. If these criteria are not fully satisfied, maintainability studies of AOP might draw artificial or inaccurate conclusion and, worse, might mislead programmers about the potential benefits and dangers of AOP mechanisms regarding software maintenance. Unfortunately, the validity and reliability of AO metrics as indicators of maintainability in AOP systems remains predominantly untested. In particular, there has been only one systematic

review on the use of metrics in AOP maintainability studies. Inspired from medical research, a systematic review is a fundamental empirical instrument based on a literature analysis that seeks to identify flaws and research gaps in existing work by focusing on explicit research questions. This paper proposes such a systematic review with the aim to pinpoint situations where existing AO metrics have been effective as surrogate measures for key maintainability attributes. In this systematic review consolidates data from a range of relevant AOP studies, highlights circumstances when the applied coupling measures are suitable to AO programs and draws attention to deficiencies where AO metrics needs to be improved.

The remainder of this paper provides Section II some background on AO programs and designs metrics. We then discuss the design of our systematic review and present its results Section III and IV. Finally, we discuss our findings in Sections V and concluded in Section VI.

II. BACKGROUND

Here we present a brief discussion on important representative AOP languages and also gives a background on metrics for AO Programs and Designs.

A. AOP Languages and Constructs

One of the reasons why the impact of AOP on maintainability is difficult to study pertains to the inherent heterogeneity of aspect-oriented mechanisms and languages. Different AOP languages tend to incarnate distinct blends of AOP and use different encapsulation and composition mechanisms. They might also borrow abstractions and composition mechanisms from other programming paradigms, such as collaboration languages (CaesarJ), feature-oriented programming (CaesarJ), and subject-oriented programming (HyperJ[14]). Most AOP languages tend to encompass conventional AOP properties such as joinpoint models, advice and aspects, or their equivalent, but each possesses unique features that make cross-language assessment difficult. Table 1 lists ten such features for AspectJ [2, 30], HyperJ [16] and CaesarJ [7], three of the most popular AOP languages (Table 1). For instance, AspectJ supports advanced dynamic pointcut designators, such as "cflow". HyperJ uses hyperspace modules to modularize crosscutting behaviour as well as non-crosscutting behaviour. HyperJ thus does not distinguish explicitly between aspects and classes in the way AspectJ does. Other abstractions unique to HyperJ include Compositions Relationships. These uses merge like operators to define how surrounding modules should be assembled. Finally, CaesarJ supports the use of virtual classes to implement a more pluggable crosscutting behaviour. This pluggable behaviour is connected with the base code through Aspect Collaboration Interfaces. In Spring AOP [24] aspects are nothing more than regular spring beans, which themselves are plain-old Java objects (POJO) registered suitably with the Spring Inversion of Control container. The core advantage in using Spring AOP is its ability to realize the aspect as a plain Java class. In Spring AOP, a join point exclusively pertains to method execution only, which could be viewed as a limitation of Spring AOP. However, in reality, it is enough to handle most common cases of implementing crosscutting concern. Spring AOP uses the AspectJ pointcut expression syntax.AspectWerkz [29] offers both power and simplicity and will help you to easily integrate AOP in both new and existing projects. AspectWerkz utilizes runtime bytecode modification to weave your classes at runtime. It hooks in and weaves classes loaded by any class loader except the bootstrap class loader. It has a rich and highly orthogonal join point model. Aspects, advices and introductions are written in plain Java and your target classes can be regular POJOs. You have the possibility to add, remove and re-structure advice as well as swapping the implementation of your introductions at runtime. Your aspects can be defined using either an XML definition file or using runtime attributes.JBoss-AOP [15, 29] allows you to apply interceptor technology and patterns to plain Java classes and Dynamic Proxies. It includes Java Class Interception, Fully compositional pointcuts caller side for methods and constructors control flow, annotations, Aspect classes, Hot-Deploy, Introductions, Dynamic Proxies and Dynamic AOP features. The PROSE system (PROSE stands for PROgrammable extenSions of sErvices) [29] is a dynamic weaving tool (allows inserting and withdrawing aspects to and from running applications) PROSE aspects are regular JAVA objects that can be sent to and be received from computers on the network. Signatures can be used to guarantee their integrity.

B. Existing AO Metrics

AO metrics aim to measure the level of interdependency between modules within a program [12], thus assessing a code's modularisation, and indirectly maintainability. This creates a challenge when designing AO metrics for AOP, as these metrics should ideally take into account each language's unique features, while still providing a fair basis for comparison multiple AOP languages. A number of AO metrics have so far been proposed for AO programs. Some are adapted from object-orientation, and transposed to account for AO mechanisms. For instance, both Ceccato and Tonella [8] and Sant'Anna et al [22] have proposed AO metrics adapted from an object-oriented (OO) metrics suite by Chidamber and Kemerer [9]. These metrics can be applied to both OO and AO programs. This is especially useful in empirical studies that perform aspect-aware refactoring. Zhao [25] uses dependency graphs to measure some AO mechanisms that are not measured individually in either Ceccato and Tonella or Sant'Anna's suites. Zhao's suite contains metrics that measure coupling sourced from AO abstractions and mechanisms independently of OO abstractions and mechanisms.

Table 1. Most popular AO abstractions and mechanisms unique to main AOP languages

AO Languages/Frameworks	Abstraction/Mechanism
AspectJ	-Intertype Declaration -Dynamic Pointcut Designators -Aspect
CeaserJ	-Aspect Collaboration Interface -Weavlet -Virtual Class
HyperJ	-Hyperspace -Concern Mapping -Hypermodule -Composition Relationship
Spring AOP	-AOP Proxy -Spring IoC -Custom Aspects
AspectWerkz	-Dynamic -Light Weight -High Performance
PROSE	-Dynamic Weaving Tool -Internet language -Signature for integration
JBoss AOP	-Dynamix Proxies -Intercepter Technology -Metadata and attribute programming

III. SYSTEMATIC REVIEW

In this section we describe the objectives and methods as well as the strategical steps carried out in the systematic review.

A. Review objectives

The objectives of this extended systematic review paper is to analyse the effectiveness of AO metrics in existing AO empirical studies as a predictor of modularity (as well as maintainability and reusability).

B. Review Strategy

We performed a systematic literature review of empirical studies of AOP based development, published in major software engineering journals and conference proceedings. Searches for papers took place in 58 papers and among them 22 popular online journal banks or were those published in International Computer Science & Engineering, IT Journals and recognised conference papers such as AOSD and ECOOP.We gave priorities to publications in conferences and relevant papers were found from ACM, SpringerLink, IEEE, Google Scholar, Online Library, and 4 were collected from other sources.

IV. RESULTS

A final set of 15 papers was finally obtained (Table 2), which is a typical sample size, for systematic reviews in software engineering [18, 28].

Table 2. Electronic Journals used for Studies

E-Journals	#Accessed	#Rejected	#Used
ACM	4	0	4
IEEExplore	4	0	4
SpringerLink	3	1	2
Elsevier	1	0	1
DOAJ	6	3	3
Conferences/ Proceedings	4	3	1
Total	22	07	15

A. Assessed Metrics Attibutes

It is difficult to select AO metrics to assess maintainability as definitions are often open to interpretations. For instance in [24], maintainability is "the ease with which a software system or component can be modified to correct faults, improve performance, or other attributes, or adapt to a changed environment". There is also no consensus about the external and internal attributes are the most significant indicators of maintainability. This is apparent in the empirical studies from the diverse selection of metrics used. Two main processes were recorded to select suitable coupling metrics. Firstly, many studies used AO metrics previously selected in similar AOP empirical studies. Secondly, results showed the Goal-Question-Metric (GQM) [6] style approach is a common technique used to select appropriate metrics in empirical studies. This approach guides researchers to: (i) define the goal of measuring maintainability, then (ii) derive external attributes that are possible indicators of maintainability, then (iii) derive from these a set of internal measurable attributes, and finally (iv) derive a set of metrics to measure the internal measurable attributes. Unfortunately, using GQM still leaves a large degree of interpretation to its users, who might independently reach divergent conclusions. One further problem with this uncertainty is that the metric selection process can become circular, especially when measuring maintainability, as external quality attributes are interconnected. For instance, stability indicates maintainability, yet maintainability can be seen as an indicator of stability.

Similar techniques for selecting appropriate metrics in empirical studies have been used in [28]. This study decided to measure attributes such as maintainability, reusability and reliability as indicators of maintainability. From this list, internal attributes such as separation of concerns, coupling, complexity, cohesion and size were selected. The final set of selected AO metrics was then defined based upon these internal attributes. We can therefore see that uncertainty on key external attributes has great impact on the remainder of the metric selection process.

This lack of conformity on these attributes has unsurprisingly affected the selected coupling metrics. For instance, maintainability is measured in studies [7, 13] through the application of 9 metrics to measure size, coupling, cohesion and separation of concerns metrics. In [9, 20, 21, 22, 23] complexity is in addition derived as an external attribute contributing to maintainability.

Similar problems have been observed in maintainability studies of object-oriented programming (OOP) this has been highlighted in a survey of existing OO empirical studies and their methodologies to predict external quality attributes [5].

Many studies acknowledge that modularity, coupling, cohesion and complexity are internal attributes that affect maintainability. Interestingly, error-proneness was the attribute that was not explicitly derived as an indicator of maintainability.

In short, different interpretations of maintainability and its subsequent derived attributes influence the AO metrics chosen or defined within the context of an empirical study. This may explain the wide range of AO metrics observed in AOP empirical studies, which we review in the next subsections.

B. AO Metrics used to Measure Maintainability

We identified 29 AO metrics in sample set of studies. A representative subset of these metrics is shown in Table 3. For each metric [29], the table lists it's name, description, and six characteristics.

Generally, the most frequent metrics were adapted from object orientation (OO).

Among them, the most common were Coupling Between Components (CBC) and Depth of Inheritance Tree (DIT), appearing in 66% of the studies. Adapted metrics hold the advantage of being based upon OO metrics that are widely used, and can be assumed reliable. The (implicit) reasoning is that adapting OO metrics to AOP maintains their usefulness. This however might not hold: DIT for instance combines both the implicit AO inheritance with the traditional OO inheritance. It thus considers two very different coupling sources together. These sources may have different affects upon maintainability and it may be beneficial to consider these seperately.

In contrast, some of the studies also use AO metrics developed for AOP, such as Coupling on Advice Execution (CAE) and Number of degree Diffusion Pointcuts (dPC). These metrics enable a more in-depth analysis of the system coupling behaviour, as they consider finer-grained langauge constructs. However, they are more likely to behave unexpectedly, being underdeveloped.

No AO metrics were found to be interchangeable, i.e. none were found to be applicable to different AO languages without any ambiguity. This is probably due to the heterogeneity of AO programming abstractions and mechanisms that makes it very hard to define metrics accurately across multiple AO languages.

The majority of metrics found in our study assess outgoing coupling connections (indicated as "Fan Out" in Table 3). This can be seen as a weakness, as both incoming and outgoing coupling connections help refactoring decisions, as discussed in [31].

C. *Measured AOP Mechanism*

OO AO metrics can be adapted to take into account AO mechanisms, producing a seemingly equivalent measure. However, this approach might miss some of specific needs of AO programs. We now review how the mechanisms of the AOP languages most commonly used in maintainability studies of AOP were accounted for in coupling measures, and draw attention to mechanisms that are frequently overlooked. Table 4 lists the mechanisms and abstractions used in the AO metrics of our study. One first challenge arises from the ambiguity of many notions. For instance, seven metrics use "modules" as their level of granularity, but what is module might vary across languages. In AspectJ an aspect may be considered a module – containing advice, pointcuts and intertype declarations, yet in CaesarJ, each advice forms its own module. More generally, many AO metrics use ambiguously terms ("module", "concern", or "component") which might be mapped to widely varying constructs in different languages. This hampers the ability of the metrics to draw cross-language comparisons [15].

Table 3. Properties of used Metrics [Burrows R et al]

Metric	Description
(DIT) Depth of Inheritance Tree [10]	Longest path from class / aspect to hierarchial root.
(RFM) Response for a Module [10]	Methods and advices potentially executed in response to a message received by a given module.
(NOC) No. of Children [10]	Immediate sub-classes / aspects of a module.
(CBC) Coupling Between Components [10]	Number of classes / aspects to which a class / aspect is coupled.
(CAE) Coupling on Advice Execution [10]	No. of aspects containing advice possibly triggered by execution of operations in a given module.
(dPC) No. of Degree Diffusion Pointcuts [31]	No. of modules depending on pointcuts defined in the module.
(InC) No. of In-Different Concerns [31]	No. of different concerns to which a module is participating.

Another challenge comes from the fact that certain phenomenon are best analysed by looking at the base and aspect codes separately. For instance, as a program evolves, it may lose its original structure. However, in AO programs, the base level and aspect level often evolve independently and have different structures. Understanding how each evolution impacts structure thus requires that each be investigated separately. This is not done in most of the empirical studies we found. We also noted that the majority of used AO metric suites did not focus on interface complexity. This is a problem as AO systems are at risk of creating complex interfaces by extracting code which is heavily dependent on the surrounding base code, and metrics are needed to identify problematic situations [22].More generally, few studies look at the connection between maintainability and specific AO mechanisms. For instance Response for a Module (RFM) measures connections from a module to methods / advices. This is useful in analysing coupling on a "per module" basis, but does not distinguish between individual AO language constructs. For instance, it adds up intertype declarations jointly with advice as they both provide functionality that insert extra code into the normal execution flow of the system. However intertype declarations differ from other types of advice as they inject new members (e.g. attributes) into the base code. AO metrics have been proposed to address this problem and measure singular mechanisms, such as advice, pointcuts, joinpoints and some intertype declarations[9,26,23], but have rarely been used in maintainability studies.

Table 4. Properties of used Metrics [Burrows R et al]

Metric	Measurement Granularity	Measurement Entity	Measurement Type	Fan In / Fan Out	Inter-changeable	AO / Adapted
(DIT)	class / aspect	class / aspect	inheritance	n/a	no	adapted
(RFM)	module	method / advice	environmental	fan out	no	adapted
(NOC)	module	class / aspect	inheritance	n/a	no	adapted
(CBC)	class / aspect	class / aspect	environmental	fan out	no	adapted
(CAE)	module	aspect	environmental	fan out	no	AO
(dPC)	module	module	environmental	fan in	no	AO
(InC)	module	concern	environmental	n/a	no	AO

Table 5. AO Metrics for Internal Software Attributes [Crystal Edge et al]

Size	Cohesion	Coupling	Separation of Concerns
Weighted Operations per Component (WOC) – number of operations (methods or advices) for each class or aspect	**Lack of Cohesion over Operations (LCO)** – the number of operation (method or advice) pairs that do not access the same attributes. This is similar to the Lack of Cohesion in Methods) LCOM object-oriented metric.	**Coupling Between Components (CBC)** – number of other components (classes or aspects) to which a class or aspect is coupled	**Concern Diffusion over Components (CDC)** – number of components (classes or aspects) whose main purpose is the implementation of a concern and the number of components that access them
Number of Attributes (NOA) – number		**Depth Inheritance Tree (DIT)** – the length	**Concern Diffusion over Operations**
Size	**Cohesion**	**Coupling**	**Separation of Concerns**
of attributes of each component (class or aspect)		of the path from a component (class or aspect) to the top of its inheritance hierarchy	**(CDO)** – number of operations (methods or advices) whose main purpose is the implementation of a concern and the number operations that access them
Lines of Code (LOC) – number of lines of code in each component (class or aspect)			**Concern Diffusion over LOC (CDLOC)** – number of transition points for each concern

To sum up, no study used metrics to measure constructs unique to AO programming languages, and very few measured finer-grained language constructs [28]. Although this depends on the particular goals of each maintainability study, this is generally problematic as each mechanism within a particular language has the potential to affect maintainability differently, and should therefore be analysed in its own right.

Metrics are useful indicators only if they have been validated. There are two complementary approaches to validate software metrics, empirical validation and theoretical validation [20, 28]. In this context, theoretical validation tests that a coupling metric is accurately measuring coupling and there is evidence that the metric can be an indirect measure of maintainability and reusability. Here we consider the 8 validity

properties suggested by Kitchenham[19]. The theoretical criteria are split into two categories: (i) properties to be addressed by all metrics; and (ii) properties to be satisfied by metrics used as indirect measures. [3] has already used the first criteria on AO metrics for AO programs. We offer some alternative viewpoints here, and also evaluate the AO metrics against properties that indirect measures should possess. When we applied this framework to the 27 AO metrics found in our review, we identified three potential violations of these criteria, discussed below.

A valid measure must obey the 'Representation Condition'. This criterion states that there should be a homomorphism between the numerical relation system and the measureable entities. In other words a coupling metric should accurately express the relationship between the parts of the system that it claims to measure. It also implies that AO metrics should be intuitive of our understanding of program coupling [20, 28]. For instance, a program with a CBC value of 6 should be more coupled than a program with a CBC value of 5. This metric holds true to its definition, however if a study is using CBC as a representation of coupling within a system this validation criteria becomes questionable. When measuring coupling we often do not perceive each connection as equal. There are different types and strengths of coupling. If we consider two AO systems; the first with 5 coupling connections via intertype declarations, and the second with 5 coupling connections via advice. Even though both systems contain 5 coupling connections, they are not equivalent, and are not equally interdependent. Various sources and types of coupling may influence the interdependency of a system in multiple ways. We found no metrics in the studies that took this finer difference into account.

Each unit of an attribute contributing to a valid measure is equivalent. We are assuming that units (modules) that are measured alongside each other are equivalent.

There are some AO metrics that only consider coupling from one language 'unit'. For example, the CAE metric satisfies this property as each connection counted by metric value involves an advice method. However, many metrics used in empirical studies of AOP assume that counting coupling connections between AO modules is equivalent to coupling connections between OO modules. As mentioned b, classes and aspects are often measured together as equivalent modules (e.g. in DIT), yet we do not have evidence that they have the same effect upon maintainability, thus violating this criteria[28].

There should be an underlying model to justify its construction. To give good reason for the creation of coupling metrics, there should be underlying evidence that the metric will be an effective indicator of maintainability. Unfortunately, this criterion definition is somewhat circular in the case of maintainability; metrics are often already constructed and applied before supporting this underlying theory and justifying their construction. In OOP it is widely accepted that there is a relationship between coupling and external quality attributes. Because AOP and OOP share similarities, we could infer that metrics that measure a specific form of coupling in OOP hold a similar potential when adapted to AOP (such as DIT, CBC). This however needs to be validated. This needs is even more acute for metrics specific to AOP (e.g. CAE), as there is less information on how coupling induced by AOP specific mechanisms correlate with maintainability.

V. DISCUSSION

Most research in AOP is focused on new design processes, languages and frameworks to support the new paradigm. However, no strong empirical evaluation was conducted to assess the effects of AOP adoption. The first step in this direction consists of defining a metrics suite for AOP software, designed so as to capture the novel features introduced by this programming style. As per this review many researchers contributed to the ongoing discussion on such metrics by distinguishing among the different kinds of coupling relationships that may exist between modules and by proposing a new metric for the crosscutting degree of an aspect (CDA). Moreover, we conducted a survey on some case studies to evaluate the information carried by the proposed metrics when applied to an OO system and to the same system migrated to AOP. Results indicate that meaningful properties, such as the proportion of the system impacted by an aspect and the amount of knowledge an aspect has of the modules it crosscuts, are captured by the proposed metrics (CDA and CIM repsectively). We visualize the definition of a common set of AOP metrics, to be adopted by the AOP community, in order to simplify the comparison of the results obtained by different research teams and to have a standard evaluation method. This paper gave a systematic review on the necessary steps for validating metrics that are to be used in an evaluation process. These steps are well-known in software engineering. The current state-of-the-art in AOSD is that one has started to work on the definition of apparently useful metrics. Now it is time to start with completing this research by providing empirical results. This will enable a larger to community to use AOSD metrics and more importantly, understand the benefits of AOSD. The results may also give hints as to for which purposes metrics extensions are useful and for which purposes separate metrics are useful [26, 27].

VI. CONCLUSIONS

We made sincere efforts in conducting the systematic review has presented valuable insights into current trends on coupling metrics measurement for AOP. This has consequently highlighted the need for fine-grained metrics that consider specific AOP constructs. We agree with the statement about existing metrics that are frequently used are therefore in danger of overlooking key contributors to AOP programs and designs [28].

We have also noticed that the AOP modularity metrics (also reusability and maintainability) studies of AOP overly concentrate on static design metrics. Dynamic AO metrics for AOP programs and designs have been applied in few of the analysed studies. This came as a surprise as many AO composition mechanisms rely on the behavioural program semantics. In fact, in this systematic review we found that it is not validated AO metrics 100% extend the OO metrics which was suggested by AOP metrics research.

VII. ACKOWLEDGMENTS

We place on records and wish to thank the author Rachel Burrows et al., for his valuable contributions to this work and for providing insight about AOP coupling metrics and systematic review of maintainability studies.

VIII. REFERENCES

[1] Arisholm, E., Briand, L., Foyen, A.: Dynamic Coupling Measurement for Object-Oriented Software. IEEE Trans. Soft. Eng. 30(8) (2004) 491-506.

[2] The AspectJ Prog. Guide, http://eclipse.org/aspectj

[3] Briand, L., Daly, J., Wüst, J.: A Unified Framework for Coupling Measurement in Object-Oriented Systems. IEEE Trans. Software Eng. 25(1) (1999) 91-121

[4] Briand, L., Wüst, J. Empirical Studies of Quality Models in Object-Oriented Systems, Advances in Computers. Academic Press (2002)

[5] Basili, V., et al.: GQM Paradigm. Comp. Encyclopedia of Soft. Eng. JW&S 1 (1994) 528-532

[6] Cacho, N. et al.: Composing design patterns: a scalability study of aspect-oriented programming. AOSD'06 (2006) 109 – 121

[7] CaesarJ homepage, http://caesarj.org

[8] Ceccato, M., Tonella P.: Measuring the Effects of Software Aspectization. WARE cd-rom (2004)

[9] Chidamber, S., Kemerer, C.: A Metrics Suite for OO Design. IEEE Trans. Soft. Eng. 20(6) (1994) 476-493

[10] Fenton, N. E., Pfleeger, S. L.: Software Metrics: a Rigorous and Practical Approach. 2nd ed. PWS Publishing Co Boston (1998)

[11] Filho, F.C., Garcia, A. and Rubira, C.M.F.: A quantitative study on the aspectization of exception handling. In Proc. ECOOP (2005)

[12] Garcia, A. et al.: Modularizing Design Patterns with Aspects: A Quantitative Study. In Proc. AOSD (2005) 3-14.

[13] Greenwood, P. et al.: On the Impact of Aspectual Decompositions on Design Stability: An Empirical Study. ECOOP (2007) 176-200

[14] Hyper/J home page, http://www.research.ibm.com/hyperspace/HyperJ.htm

[15] JBoss AOP, http://labs.jboss.com/jbossaop

[16] Kastner, C., Apel, S., and Batory, D.: Case Study Implementing Features Using AspectJ. In Proc. SPLC (2007) 223-232.

[17] Kitchenham, B., et al.: Systematic Literature Reviews in Software Engineering – A Systematic Literature Review. Information and Software Technology (2008)

[18] Kitchenham, B.: Procedures for Performing Systematic Reviews. Joint Tech. Rep. S.E.G.(2004)

[19] Kitchenham, B., Pfleeger, S.L., & Fenton, N.: Towards a Framework for Software Validation Measures. IEEE TSE, 21(12) (1995) 929-944

[20] Kulesza, U. et al.: Quantifying the Effects of Aspect-Oriented Programming: A Maintenance Study. In Proc. ICSM (2006) 223-233

[21] Marchetto, A..: A Concerns-based Metrics Suite for Web Applications. INFOCOMP journal of computer science 4 (3) (2004)

[22] Sant'Anna, C. et al.: On the Reuse and Maintenance of Aspect-Oriented Software: An Assessment Framework. In Proc. SBES (2003) 19-34

[23] Sant'Anna, C. et al.: On the Modularity of Software Architectures: A Concern-Driven Measurement Framework. In Proc. ECSA (2008)

[24] Spring AOP, http://www.springframework.org

[25] Zhao, J.: Measuring Coupling in Aspect-Oriented Systems. Int. Soft. Metrics Symp. (2004)

[26] Zimmermann, T., Nagappan, N.: Predicting defects using network analysis on dependency graphs. ICSE (2008) 531-540

[27] K.G.Kouskouras, A.Chatzigeorgiou and G. Stephanides: Facilitating software extension with design patterns and Aspect-Oriented-Programming, http://www.sciencedirect.com/science

[28] Burrows R,Garcia A, et. al, Coupling Metrics for Aspect-Oriented Programming-A Systematic Review of Maintainability Studies(Extended Version) Proceedings of the 4th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2009), 9-10 May 2009, Milan, Italy (12p)

[29] Open Source Aspect-Oriented Frameworks in Java http://java-source.net/open-source/aspect-orientedframeworks

[30] Laddad, R. 'Enterprise AOP with Spring Applications: AspectJ in Action' 2nd Edition, Manning Publications, 2009, Manning Publications

[31] Kiczales, G. et al.: Aspect-Oriented Programming. ECOOP (1997) 220-242

[32] Sirbi. K et al., Metrics for Aspect Oriented Programming-An Empirical Study, International Journal of Computer Applications (0975 – 8887), Volume 5– No.12, August 2010.

