2009 International Conference on Computer Technology and Development

A Coupling and Cohesion Metrics Suite for Object-Oriented Software

Sukainah Husein[1], Alan Oxley[2]
Computer and Information Science Department
Universiti Teknologi PETRONAS, UTP
Bandar Seri Iskandar, Malaysia
sukainah.husein@gmail.com[1], alanoxley@petronas.com.my[2]

***Abstract* — The increasing need for software quality measurements has led to extensive research into software metrics and the development of software metric tools. To maintain high quality software, developers need to strive for a low-coupled and highly cohesive design. However, as mentioned by many researchers, coupling and cohesion metrics lack formal and standardized definitions and thus for each metric there is more than one interpretation. This paper introduces our view of coupling and cohesion metrics and our implementation approach. Coupling and cohesion metrics are calculated by considering a number of relationships, which were introduced by several researchers. Based on the relationships, some sets of metrics were chosen. The selected metrics are then formalized to aid the implementation of the metrics to actual software codes. The formalized metrics have been implemented in CCMETRICS, a tool for calculating the coupling and cohesion metrics of object-oriented software.**

Keywords —coupling, cohesion, object-oriented, software metrics.

I. INTRODUCTION

The term "coupling" was first used in software engineering by Stevens et al in the days when structured programming was the norm. It was defined as "the measure of the strength of association established by a connection from one module to another" [1]. In the context of object-oriented design, coupling is seen in terms of how one class is connected to another. Coupling indicates the dependency of a class on the other class. High dependency may decrease reusability of the class and increase maintenance effort. Changes in one class might affect dependent classes and thus lead to a ripple effect [1].

Coupling between classes can be reduced by promoting cohesiveness of the class [1]. A class is cohesive if the association of elements declared in the class is focused on accomplishing a single task. Furthermore, humans use problem decomposition to make a complex problem easier to handle. However, there is a limit to how far one can decompose before the effort required to integrate, and later maintain, becomes prohibitive. This latter issue will not be discussed as it is outside the scope of this paper.

Coupling and cohesion need to be automatically measured to reduce the measurement effort, subjectivity, and possible errors. Therefore, coupling and cohesion metrics were introduced ranging from syntactic to semantic analysis [2-4], and software metrics tools were developed e.g. VizzAnalyzer, Analyst4j, Understand, and Ckjm. A study on comparing these tools was carried out by [5]. However, implementation of coupling and cohesion metrics from these tools are not enough for coupling and cohesion investigation due to several relationships being overlooked. In this paper, we tried to look at the relationships that make up coupling and cohesion by examining existing frameworks and selecting metrics that correlate to the respective relationships. The selected metrics are then formalized to show how these metrics are implemented.

This paper is divided into five main parts. The first part is the introduction section. The second part is the Coupling and Cohesion Framework section that explains the framework used in the implementation. The third section briefly describes the metrics selected for the tool and the formal definition of the metrics. Next, section four gives a general description of the tool. Finally, section five discusses the unresolved disputes and suggests future work.

II. COUPLING AND COHESION FRAMEWORK

Coupling and cohesion relate to particular relationships that exist between classes, and within a class, respectively. Relationships that contribute to coupling were defined by Eder et al [6]. Three types of coupling were defined: interaction coupling, component coupling and inheritance coupling.

Interaction coupling refers to the existence of a relationship caused by message passing and method invocation [6].

Component coupling refers to the existence of a relationship caused by abstract data types. In an abstract data type, the attribute of a class is declared as a non-primitive type. An abstract data type (ADT) can be declared in one of three ways - as an attribute of a class (class field), an attribute of a method, or a method parameter [6].

Inheritance coupling refers to the existence of a relationship caused by inheritance or if one class is an ancestor of another class [6].

Another framework was introduced by Briand et al [7] who suggested that the existence of data abstraction in method return type also leads to coupling.

Based on the above frameworks, the consolidated relationships are summarized in Table 1, where mA_i is a method of class A; vA_i is an attribute of class A. To illustrate the relationships, we will use the following assumption in this paper. Let $E(c)$ be the set of elements of a class c;

978-0-7695-3892-1/09 $26.00 © 2009 IEEE
DOI 10.1109/ICCTD.2009.209

$$E(c) = m(c) \cup v(c)$$

where $m(c)$ is the set of methods of class c and $v(c)$ is the set of attributes of class c.

$$m(c) = \{mc_1, mc_2, mc_3, \ldots, mc_k\}$$

$$v(c) = \{vc_1, vc_2, vc_3, \ldots, vc_i\}$$

TABLE 1: CONSOLIDATED RELATIONSHIPS FOR COUPLING

Classification	Relationship	Description	Name
Method Invocation Coupling	Method invocation	mA_i is invoked by mB_k	MI
Global Data Coupling	Sharing of data	vA_i is invoked by vB_i	AR
Data Abstraction Coupling	Attribute of class or method as ADT	B is the type of vA_i	DA
	Method parameter as ADT	B is the type of formal parameter of mA_i	MP
	Method return type as ADT	B is a return type of mA_i	MR
Inheritance Coupling	Inheritance	Class A is the ancestor of class B	IH

Cohesion framework introduced by Eder et al [6], provides a qualitative list that defines the degree of cohesion in an object-oriented environment. The framework does not explicitly mention the types of relationships that constitute cohesion, thus it is not applicable in the structured syntactic cohesion measurement.

Our approach is based on one by Briand et al whose framework is one of the few that mentions relationships relevant to cohesion. Briand et al defines two types of cohesion, Data-Data interaction (DD-interaction) and Data-Method interaction (DM-interaction) [3],[8].

DD-interaction refers to the existence of a relationship caused by interactions between data declaration. Here, data declarations refer to local and global type abstract data, the class itself, and global attributes [3].

DM-interaction refers to the existence of a relationship caused by interactions between data, one or both of which are at the method level. Method level attributes include local attributes, method parameters, and method return types [3].

Briand et al's framework is more applicable for a high level design measurement [8]. Thus, the framework does not include methods' attributes because the method body is not visible at that stage. We have attempted to fill this gap in our cohesion metrics approach by extending the framework that defines the type of relationships for cohesion at low level design (given the source codes of the software). Thus, in our definition, data declaration also includes methods' attributes.

Framework by Hitz and Montazeri [9] suggested the inclusion of local method invocation to remedy the problem with access methods (set/get methods primarily in Java). Thus, we included this mechanism in the consolidated framework for cohesion.

These relationships are summarized in Table 2.

TABLE 2: CONSOLIDATED RELATIONSHIPS FOR COHESION

Classification	Relationship	Description	Name
DD-interaction	Global attribute	vA_i is related/used by vA_i	AR
DM-interaction	Local attribute	vA_i is related/used by vA_j declared in mA_k	AR
	Method parameter	A is declared as formal parameter of mA_k	MP
	Method return type	A is declared as return type of mA_k	MR
MM-interaction	Method invocation	mA_i is invoked by mA_j	MI

III. METRICS SELECTION

This section of the paper discusses the metrics chosen based on the relationships that have been defined in the previous section. In most cases, we need more than one metric in order to satisfy all types of relationships. Each metric can represent for more than one relationship, however it is preferable to have a metric that only represents one type of relationship. Coupling metrics, defined here, indicate how dependent a class is on others. The higher the metric value, the more dependent the class is. This means that the class greatly uses, and thus relies on, other classes to complete its tasks. While for cohesive metrics, a high metric value indicates high cohesion which indicates good software design or programming practice.

A. Coupling Metrics

Data Abstraction Coupling (DAC) was introduced by Li & Henry. It was originally defined as "the number of abstract data types (ADTs) defined in a class" [10]. This corresponds to part of DA in Table 1. In our opinion, this definition of the metric does not go far enough as attributes of abstract data can be declared as not only class fields but also as local method attributes.

Message Passing Coupling (MPC) was originally defined as the "number of send statements defined in a class" [10]. It means that the degree of coupling is affected by the extent to which a class engages in message passing [2]. Message passing can be achieved when a method is invoked as an expression (calling an external method in local method) or when actual parameters are passed. This corresponds to MI and AR in Table 1.

Class-Method Interaction (CM) was introduced by Briand et al in their coupling measures suite [2]. CM interaction occurs when a method in a class has another class as its parameter or return type. These relationships correspond to MP and MR respectively in Table 1.

Global Coupling (GC) was introduced by Offut et al [11]. It occurs when a class is accessing public attributes belonging to other classes. This is a strong type of coupling because it may violate the concepts of encapsulation and information hiding if the identifier of the attributes is not carefully designed, thus this type of coupling should be kept to a minimum. These relationships correspond to AR in Table 1.

Inheritance coupling (IC) Offut et al and needs no explanation [11]. It is also simple to calculate as one just looks for the number of instances in which inheritance occurs. In Java, for example, inheritance is identified through the keywords 'extends' and 'implements'[11]. These relationships correspond to IH in Table 1.

B. Cohesion Metrics

Ratio of Cohesive Interaction (RCI) was introduced by Briand et al based on DD- and DM- interactions [3]. It is defined as the total of actual interactions divided by the maximum possible interactions in the class. Originally, RCI [3] was formalized as:

$$RCI(c) = \frac{|CI(c)|}{|MaxI(c)|}$$

where $CI(c)$ is the set of cohesion interactions of class c based on actual DD- and DM- interactions and $MaxI(c)$ is the set of all possible DD- and DM- interactions of class c. Again, DD- and DM- interactions mentioned in [3] are more applicable for high level design measurement. Thus, in our implementation we made an adjustment so that methods' local attributes are included as data.

IV. Formal Definitions of Selected Metrics

For implementation purposes, we formalize the metrics definitions. This aids in understanding how to apply the metrics to actual software source code.

A. Coupling Metrics

DAC was originally defined as the number of class fields [10]. We have redefined it to include both the number of class fields and the number of method attributes. With this new definition, DAC now corresponds to DA in Table 1. Let:

$$DAC(c) = |DAF(c)| + |DAA(c)| \quad (1)$$

where *DAF(c)* is the set of abstract class fields and *DAA(c)* is the set of abstract method attributes.

CM corresponds to MP and MR in Table 1. Once again we have redefined the original metric. The redefined metric involves the number of method formal parameters and method return types for abstract data types. With this new definition, let:

$$CM(c) = |MP(c)| + |MR(c)| \quad (2)$$

where *MP(c)* is the set of abstract method formal parameters and *MR(c)* is the set of abstract return types.

GC satisfies AR types of relationship shown in Table 1. It too should be redefined here. The redefined metric involves the number of connections that engage with other classes' fields. This engagement might occur in several ways. This includes static and object referenced attributes. Let:

$$GC(c) = |ARE(c)| + |ARP(c)| \quad (3)$$

where *ARE(c)* is the set of external attributes used as expressions and *ARP(c)* is the set of external attributes used as actual parameters.

MPC satisfies MI type of relationship. In the definition by [2], it only includes static method invocations. However, in our implementation we have redefined it to include both static and methods referenced by objects of the class. Method invocations might occur in numbers of ways as defined below. Let:

$$MPC(c) = |MIE(c)| + |MIP(c)| \quad (4)$$

where *MIE(c)* is the set of external method invoked as expression and *MIP(c)* is the set of external method invoked as actual parameter.

IC satisfies IH type of relationship and is defined as the number of parent classes. Let:

$$IC(c) = |IH(c)| \quad (5)$$

where *IH(c)* is the set of parent classes.

B. Cohesion Metrics

Based on Table 2, method invocation is also regarded as possible cohesion relationship. Thus, we adopted RCI [3] and extend the definition to cater for MM-interaction. As mentioned earlier, RCI [3] is the ratio of the actual number of interactions in a class with the total possible interactions.

$|CI(c)|$ is the sum of the total DD-interactions, DM-interactions, and MM-interaction.

$$|CI(c)| = |DD(c)| + |DM(c)| + |MM(c)| \quad (6)$$

where $|DD(c)|$ is the number of connections that occur between variables in class c, $|DM(c)|$ is the number of connections that occur between methods and variables in a class c and $|MM(c)|$ is the number of connections that occur between methods and methods in class c. Note that $|DD(c)|$, $|DM(c)|$ and $|MM(c)|$ are not vectors, so we ignore the locus or direction of the connections.

To calculate $|DD(c)|$ and $|DM(c)|$, we first have to define what are considered as variables of class c. Let us recall

that $v(c)$ is the set of attributes. In our implementation, these attributes are expected to be found in several roles. Hence, we can subdivide this set as follows:

VAF(c) is the set of variables declared as class fields, *VAA(c)* is the set of variables declared as method attributes, *VMP(c)* is the set of variables declared as method formal parameters. Thus,

$$v(c) = VAF(c) \cup VAA(c) \cup VMP(c) \quad (7)$$

To simplify the calculation, we include the class c itself (as an implicit, public type [3]) in class field. *MaxI*(c) can be thought of in terms of combinatorics and products.

$$MaxI(c) = {}^{k}C_2 + {}^{x}C_2 + (x*(y+z)) \quad (8)$$

where x is the size of *VAF(c)*, y is the size of *VAA(c)*, z is the size of *VMP(c),* and k is the number of methods is class c.

IV. CCMETRICS Software System

This section briefly describes CCMETRICS, a tool we have developed for calculating coupling and cohesion metrics in object-oriented software.

A *Program Flow*

The program flow of CCMETRICS is summarized in Figure 1. The program receives object-oriented software source code as input and then it is parsed according to the language. The parser is based on grammar files. Another Tool for Language Recognition (ANTLR) was used to construct these files. The parsed input is then reduced by extracting keywords according to certain rules which are adopted from the formalized definition in section 3. The rules are similar to those that make up a grammar but in our case the rules define the relationships relevant to coupling and cohesion. If the certain criteria are met, CCMETRICS computes the final metrics values and displays the results.

B. *User Interface*

CCMETRICS allows users to evaluate source codes on package and class level. On package level, metrics of all classes are evaluated and summed together to get package level metrics. The user interface for CCMETRICS is shown in Figure 2.

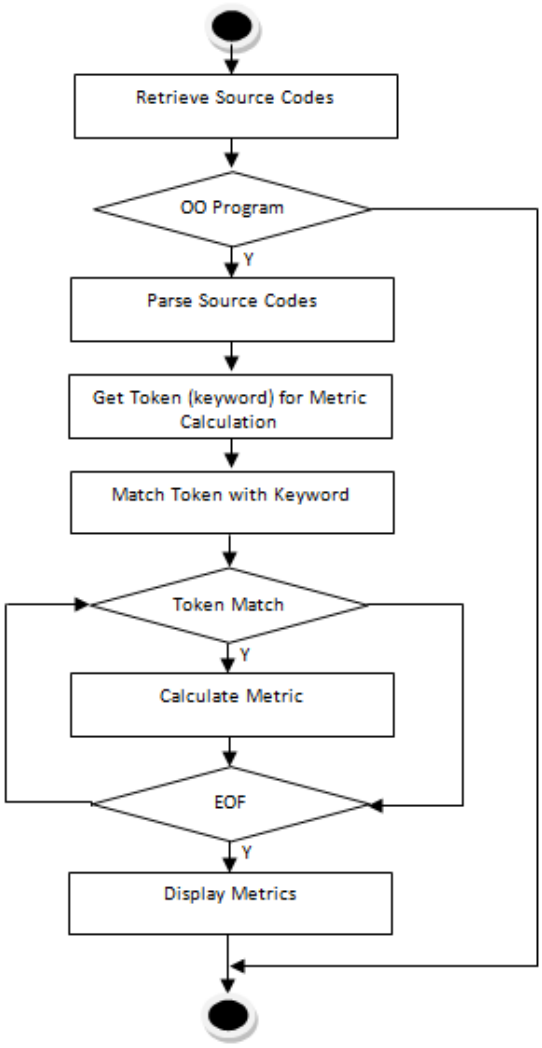
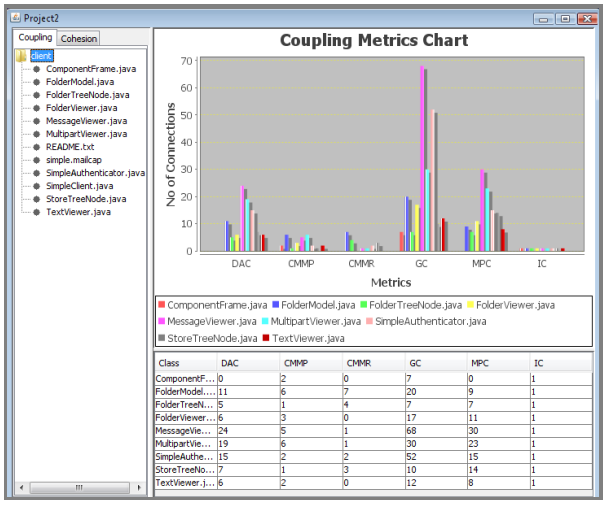


Figure 1. CCMETRICS Program Flow



Class	DAC	CMMP	CMMR	GC	MPC	IC
ComponentF…	0	2	0	7	0	1
FolderModel…	11	6	7	20	9	1
FolderTreeN…	5	1	4	7	7	1
FolderViewer…	6	3	0	17	11	1
MessageVie…	24	5	1	68	30	1
MultipartVie…	19	6	1	30	23	1
SimpleAuthe…	15	2	2	52	15	1
StoreTreeNo…	7	1	3	10	14	1
TextViewer.j…	6	2	0	12	8	1

Figure 2. CCMETRICS User Interface

V. Conclusion and Further Work

Coupling between classes exists because one class is relying on the other. There are several possible relationships that are usually implemented in software that affect the degree of coupling of a class. With cohesion, relationships between elements in a class are desired. Although cohesion is difficult to define, there are several relationships that represent the degree of cohesiveness of a class. Based on the relationships for both coupling and cohesion, metrics were evaluated and chosen. CCMETRICS, a tool for computing coupling and cohesion metrics of object-oriented software has been developed.

In future, we hope to be able to further investigate the extent to which each of the different types of relationship contributes to cohesion and coupling. The current version of CCMETRICS handles source code. The next version will calculate coupling and cohesion metrics for UML representations of software. Following this, the importance to software engineers of having a tool that calculates metrics for both UML and source code representations will be investigated.

References

[1] W. P. Stevens, G. J. Myers, and L. L. Constantine, "Structured design," *IBM Systems Journal,* vol. 13, pp. 115–139, 1974.

[2] L. C. Briand, J. W. Daly, and J. Wu¨st, "A unified framework for coupling measurement in object oriented systems," *IEEE Transactions on Software Engineering,* vol. 25, pp. 91–121, 1999.

[3] L. C. Briand, J. W. Daly, and J. Wu¨st, "A unified framework for cohesion measurement in object oriented systems," *Empirical Software Eng.: An Int'l Journal,* vol. 3, pp. 65-117, 1998.

[4] A. Marcus, D. Poshyvanyk, and R. Ferenc, "Using the Conceptual Cohesion of Classes for Fault Prediction in Object-Oriented Systems," *IEEE Transactions on Software Engineering,* vol. 34, 2008.

[5] R. Lincke, J. Lundberg, and W. Löwe, "Comparing Software Metrics Tools," in *Proceeding of the 2008 International Symposium on Software Testing and Analysis*, 2008, pp. 131-142.

[6] J. Eder, G. Kappel, and M. Schrefl, "Coupling and cohesion in object-oriented systems," University of Klagenfurt, Technical report 1994.

[7] L. C. Briand, P. Devanbu, and W. Melo, "An Investigation into Coupling Measures for C++," in *Proc. 19th Int'l Conf. Software Eng., ICSE '97*, 1997, pp. 412-421.

[8] L. C. Briand, S. Morasca, and V. R. Basili, "Defining and Validating Measures for Object-Based High-Level Design," *IEEE Transactions on Software Engineering,* vol. 25, pp. 722-743, 1999.

[9] M. Hitz and B. Montazeri, "Measuring coupling and cohesion in object-oriented systems," in *Proceedings of the International Symposium on Applied Corporate Computing*, 1995, pp. 412–421.

[10] W. Li and S. M. Henry, "Object-Oriented Metrics that Predict Maintainability," *Journal of Systems and Software,* vol. 23, pp. 111-122, 1993.

[11] J. Offutt, A. Abdurazik, and S. R. Schach, "Quantitatively measuring object-oriented couplings," *Software Quality Journal,* vol. 16, pp. 489-512, 2008.