Design based Object-Oriented Metrics to Measure Coupling and Cohesion

K.G.S.VENKATESAN
ASSOCIATE PROFESSOR
DEPT OF CSE
BHARATH UNIVERSITY CHENNAI-600073
TAMILNADU,INDIA
M.ELAMURUGASELVAM
PG SCHOLAR
DEPT OF CSE
BHARATH UNIVERSITY CHENNAI-600073
TAMILNADU,INDIA.
Email:elamurugaselvam@gmail.com
Email:venkatesh.kgs@gmail.com

Abstract:

The object oriented design and object oriented development environment are currently popular in software organizations due to the object oriented programming languages. As the object oriented technology enters into software organizations, it has created new challenges for the companies which used only product metrics as a tool for monitoring, controlling and maintaining the software product. This paper presents the new object oriented metrics namely for coupling of class by counting the number of associated classes within a class & total associated class and cohesion at the method and function level for cohesion to estimates object oriented software. In order to this, we discuss in this paper object oriented issues and measures with analysis of object metrics through coupling and cohesion to check the complexity with weight count method. We also discuses the estimation process after analysis of proposed object oriented metrics to measures and check the better performance of object oriented metrics in comparison to other object oriented metrics.

Key Words: Object oriented metrics, coupling, cohesion and complexity

1. Introduction

In the era of software development, many companies have started to introduce object oriented technology into their software development environment. The object oriented approach to software development facilitates and encourages programming practice that increases reusability, correctness and maintainability in the code. The object oriented design, object oriented language and object oriented development environment are currently popular in software organizations. As the object oriented technology enters into software organizations, it has created new challenges for the companies which used only product metrics as a tool for monitoring, controlling and maintaining the software product. Therefore, metrics which reflect the specificities of object oriented paradigm must be defined and validate in order to be used in software organization. Some studies have conducted that functional product metric are not sufficient for characterization accessing and predicting the quality of object oriented software systems. For example based on the study of McCabe Cyclomatic Complexity appears to be an inadequate metric for use in software development. The advantage of object oriented and other modern software engineering technique are offset by the continuing increases in the size and complexity of software system and project. A wide range of metrics can aid you in measuring and increases reusability, correctness, maintainability in the code. Object oriented design and development are popular concepts in today's software development environment. Object oriented software development requires a different approach from functional development methods, including metrics used to evaluate the software. Functional metrics and data design metrics measure the design structure or data structure independently. How ever, object oriented metrics must be able to treat function and data as combined, integrated object. Object oriented metrics are different due to the different approach in program paradigm and in object oriented language itself. Object oriented metrics are used to evaluate the following attributes namely efficiency, complexity, understandability, reusability, testability and maintainability.

3. Object-Oriented Metrics

Increasingly, object-oriented measurements are being used to evaluate and predict the quality of software [1]. A growing body of empirical results supports the theoretical validity of these metrics [2,3]. The validation of these metrics requires convincingly demonstrating that the metric measures what it purports to measure and the metric is associated with an important external metric, such as reliability, maintainability and fault-proneness [4]. Often these metrics have been used as an early indicator of these externally visible attributes, because the externally visible attributes could not be measures until too late in the software development process. Object oriented metrics evaluate the object oriented concept: methods, classes, cohesion, coupling and inheritance. Object oriented metrics focus on the internal object structure. Object oriented metrics measure externally the interaction among the entities. Object oriented metrics measures the efficiency of an algorithm. Object oriented metrics are primarily applied to the concepts of classes, coupling, and inheritance. Preceding each metric, a brief description of the object oriented structure is given. A class is a template from which objects can be created. This set of objects shares a common structure and a common behavior manifested by the set of methods. A method is an operation upon an object and is defined in the class declaration. A message is a request that an object makes of another object to perform an operation. The operation executed as a result of receiving a message is called a method. Cohesion is the degree to which methods within a class are related to one another and work together to provide well-bounded behavior. Effective object oriented designs maximize cohesion because cohesion promotes encapsulation. Coupling is a measure of the strength of association established by

a connection from one entity to another. Classes are coupled when a message is passed between objects; when methods declared in one class use methods or attributes of another class.

Chidamber and Kemerer's metrics suite: They have defined six metrics for the oo design.
Weighted method per class (WMC): It is defined as the sum of complexities of all methods of a class.
Depth of inheritance tree (DIT): It is defined as the maximum length from the node to the root of the tree.
Number of children (NOC): It is defined as the number of immediate subclasses.
Coupling between object (CBO): Two classes are coupled when methods declared in one class use methods or
instance variables of the other class.
Response for a class (RFC): It is defined as number of methods in the set of all methods that can be invoked in
response to a message sent to an object of a class.
Lack of cohesion in methods (LCOM): It is defined as the number of different methods within a class that reference a given instance variable.
MOOD Suite: This set includes the following metrics

4. Proposed Object Oriented Metrics for Coupling and Cohesion

We proposed here the specific measures for coupling and cohesion to measure the object oriented application that can be computed.

4.1 Coupling

In 1974, Stevens et al. first defined coupling in the context of structured development as "the measure of the strength of association established by a connection from one module to another [7]. "Coupling is a measure of of two objects. For example, objects X and Y are coupled if a method of object X calls a method or accesses a variable in object Y. Classes are coupled when methods declared in one class use methods or attributes of the other classes. The Coupling Factor (CF) is evaluated as a fraction. The numerator represents the number of non-inheritance couplings. The denominator is the maximum number of couplings in a system. The maximum number of couplings includes both inheritance and non-inheritance related coupling. Inheritance based couplings arise as derived classes (subclasses) inherit methods and attributes form its base class. The CF metric is included in the MOOD metric suite. Empirical evidence supports the benefits of low coupling between objects [8, 9, 10]. The main arguments are that the stronger the coupling between software artifacts, the more difficult it is to understand individual artifacts, and hence to correctly maintain or enhance them; the larger the sensitivity of (unexpected) change and defect propagation effects across artifacts; and consequently, the more testing required to achieve satisfactory reliability levels. Additionally, excessive coupling between objects is detrimental to modular design and prevents reuse. To summarize, low coupling is desirable. To calculate the coupling caused by association relations we have proposed the following object oriented metrics:

• Number of associated classes within a class (NAC): This metric gives the number of associated classes

with a particular class. All kind of associations (e.g. association, aggregation and composition) will be used to count the number of associated classes.

• Total Associated Class (TAC): This metric gives the number of times all associated attributes of a particular class type are used by methods of a user class.

4.1.1 Class coupling

We have found four types of coupling among the classes namely, dependency, inheritance, association and performance. A class in OO implementation may have all such types of coupling with other classes. This observation makes the coupling a three dimension function that can be expressed by something like following equation.

Coupling Class = a*Ac* + b*Db* + c*Ic* + d*Pd*

S. No. Coefficients Language C++ Language Java

1 **a*Ac*** 2 No Association

2 **b*Db*** 3 4

3 **c*Ic*** 5 0

4 **d*Pd*** 5 4

C++ = 2+3+5+5 = 15

JAVA = 0+4+0+4 = 8

Where a, b, c and d are the coefficients of unknown values and *Ac*, *Db*, c*Ic and* d*Pd* are representing association, dependency, inheritance and performance types of coupling respectively. We are also not sure about the degree of stated equation. Therefore, we are unable to form an equation based on four dimensions of coupling. We can imagine intuitively all dimensions of coupling have a different contribution to over class coupling. But the weights of their contributions by the types of coupling are undetermined. Undetermined weights also hinder us to define an ordinal scale classification for measuring the overall class coupling.

4.2 Cohesion

Cohesion refers to how closely the operations in a class are related to each other. Cohesion of a class is the degree to which the local methods are related to the local instance variables in the class. The CK metrics suite examines the Lack of Cohesion (LCOM), which is the number of disjoint/non-intersection sets of local methods

[11]. There are at least two different ways of measuring cohesion:

1. Calculate for data, function and method field in a class to calculate what percentage of the methods and

function use that data field. Average the percentages then subtract from 100%. Lower percentages mean

greater cohesion of data, function and methods in the class.

2. Methods and functions are more similar if they operate on the same attributes. Count the number of disjoint sets produced from the intersection of the sets of attributes used by the methods. High

cohesion indicates good class subdivision. Lack of cohesion or low cohesion increases complexity, thereby increasing the likelihood of errors during the development process. Classes with low

cohesion could probably be subdivided into two or more subclasses with increased cohesion. This proposed metric evaluates the design implementation as well as reusability.

Weight of Class C = Number of Method and Function in a Class / Total number of public method and function. Every subset tree in a class contributes to overall functionality of the class and every method in the subset tree contributes to overall functionality presented by the subset tree, but such contribution may not be equal on both levels. If we just consider the contribution of method for a subset tree, we can argue that method which has highest S(MF) value in the subset tree is the most related method and function to the overall functionality presented by the subset tree. Based on this argument, we redefine the relatedness of method as a relative measure to maximum S(MF) for method and function in subset tree. Therefore, we have divided the S(MF) for a method MF by the maximum S(MF) for a method and function in a given subset tree. The formula for measuring the relatedness of public methods and function can be defined using following equation.

R (MF) = S(MF) / Maximum of S(MF) for all methods and function For measuring the cohesion for the subset tree, we will calculate the average measurement of relatedness R(MF) of all methods in the subset tree.

Cohesion for Subset Tree = Σ R(MF) / TMF where TMF stands for the total number of methods and function in subset tree.

5. Analysis and result of proposed metrics:

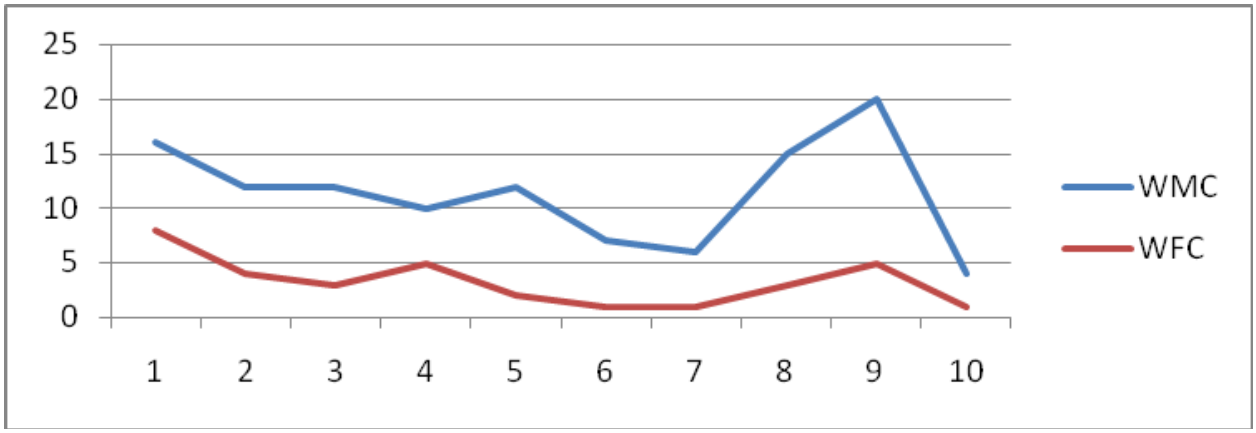
Firstly, we analyze the coupling metrics. Here, two metrics have proposed namely Number of associated class within a class (NAC) and Total associated class (TAC). According the NAC number of associated classes with a particular class are measured. All kind of associations are used to count the number of associated class. With the help of the graphical representation, we analyze the coupling for NAC. For example:

Fig. 1. Example of Aggregation

Here, we take the example of aggregation. According to this, the coupling value is 4 because we are taking blade, engine, wheel and deck as individual module which are associated with lawnmower. And when we Lawn Mower Blade Engine Wheel Deck

compare the proposed metrics with Coupling between Object (CBO), then the coupling value remain same. But the key difference between these two is that the proposed metric uses all kind of association i.e. association, aggregation and composition in comparison of Coupling between Object (CBO). Thus, we can say that the proposed metric NAC is better than CBO. And, according to Total Associated Class (TAC) we can measure the number of times all associated attributes of a particular class type are used by methods of a user class. Here the WMC is also less because when a same method is called again and again then that has been counted as a single method. After the coupling metrics, cohesion metric is analyzed. The proposed metric Weight Factor of Class (WFC) evaluate the design implementation. According to this metric, we measure the weight of a class and on the basis of that weight, we can analyze the complexity. We compare our proposed metric with Weighted Method per Class (WMC) to

check the complexity. To empirically validate this metric, we are applying both of these on some program and get the result. For this, a table is drawn which consist the values for WFC and WMC. This is the following table:



The chart shows that how complexity is reduced with proposed metric i.e. WFC. We can also measure the complexity of a subtree where inheritance is used. But for this, firstly we calculate the relatedness and then cohesion of subtree. We can measure the relatedness of public methods and function using following equation:

R (MF) = S(MF) / Maximum of S(MF) for all methods and function

6. Estimation and Complexity of Object Oriented System

Our proposed metrics are helpful in calculating the Weighted Methods per Class (WMC) which is a count of the methods implemented within a class or the sum of the complexities of the methods. The measurement isdifficult to implement since not all methods are assessable within the class hierarchy due to inheritance. The number of methods and the complexity of the methods involved is a predictor of how much time and effort is required to develop and maintain the class. The larger the number of methods in a class, the greater the potential impact on children; children inherit all of the methods defined in the parent class. Classes with large numbers of methods are likely to be more application specific, limiting the possibility of reuse. WMC measures the complexity of an individual class. A class with more member functions than its peers is considered to be more complex and therefore more error prone [2]. The larger the number of methods in a class, the greater the potential impact on children since children will inherit all the methods defined in a class. Classes with large numbers of methods are likely to be more application specific, limiting the possibility of reuse. This reasoning indicates that a smaller number of methods and functions are good for usability and reusability. However, more recently, software development trends support have more, smaller methods over fewer, larger methods for reduced complexity, increased readability, and improved understanding [12]. This reasoning contradicts the prior reasoning of the last paragraph. The most recent recommendation (of more, smaller methods) is most likely more prudent. However, if a method is in a large inheritance tree having a large number of methods may not be advisable. Often, the WMC calculation considers complexity and the count of the number of methods applies a weighted

complexity factor [11]. Poor estimates and misconceptions about the estimating process often contribute to the failure of software projects. Many of these have been exacerbated by the transition to Object Oriented methods. This technical report attempts to address as many of them as possible.

Lack of historical data on which estimates are to be based

Since OO methods are relatively new, few organizations have a significant amount of data from these projects.

Lack of estimating knowledge

Only time can solve this. As managers get more experience in estimating Object Oriented projects they will get better at it.

Lack of a systematic inference process

Sound techniques or models suited to object-oriented development. While the available Object Oriented estimation techniques may not be widely known, several useful techniques have been developed.Every estimation model makes assumptions about the scope covered by the estimates. The Object Oriented development model differs in significant ways from structured development models. Misunderstanding the capabilities of Object Oriented methods leads to infeasible plans. Aggressive claims have been made for the levels of productivity, quality, and reuse. Data compiled in failure to recognize and address the uncertainty inherent in software estimates. The increased uncertainty due to the factors discussed above makes understanding the confidence associated with an estimate more important for Object Oriented development. Some specific factors to consider in selecting an estimation approach include range of activities covered by the model or approach. Does it match the activities to be estimated for the project under study? Documentation, requirements analysis, and final qualification testing are some of the activities that models often fail to include, Range of activities covered by the model or approach. Does the approach address estimating size, effort, schedule, quality, and reuse? Does it provide guidance for periodic re-estimation of these dimensions? Is the model a theoretical exercise, or has it been proven in software engineering situations? Consider the depth of data, number of users/evaluators, and robustness of the evaluation methods. Availability of appropriate local historical data to calibrate the model or approach. Is data available from past projects for the specific parameters used in the model? Summarizes relevant data from several sources. Ability to capture the necessary data from the current project. Does the Object Oriented development method used produce the artifacts, such as "key classes," that get counted to drive the model? Ability to approximate input parameters. Is there a reasonable method for approximating the values of parameters for which no historical data is available? A sample of software might be examined in detail, or data from a commercial database could be used to establish a starting point. Availability, prices, documentation, and support for estimation tools that implement the model, if any.

7. Conclusion

The object oriented metrics are supported by most literature and some object oriented tools. The need for such metrics is particularly acute when an organization is adopting a new technology for which established practices have yet to be developed. These new metrics evaluate the concept of object oriented methods, classes at coupling and cohesion level for feedback which is useful for software engineers, designers and managers. We present the analysis of object oriented metrics and measures with new proposed metrics to estimate the process especially for object oriented software systems for comparison. Most of these metrics can be useful quality indicators and most of them are

complementary indicators which are relatively independent from each other. The object oriented metrics seem to be better in object oriented in comparison to functional metrics. This paper provides motivation for further investigation and refinement of object oriented metrics and measures with estimation process.

8. References

[1] El Emam, K., (2000). "A Methodology for Validating Software Product Metrics," National Research Council of Canada, Ottawa, Ontario, Canada NCR/ERC-1076.
[2] Basili, V.R.; Briand, L.C.; Melo, W. (1996). "A Validation of Object Orient Design Metrics as Quality Indicators," *IEEE Transactions on Software Engineering*, vol. 21, pp. 751-761.
[3] Fenton, N.E.; Pfleeger, S.L. (1998). *Software Metrics: A Rigorous and Practical Approach*: Brooks/Cole Pub Co.
[4] Briand, L.C.; Daly, J.W.; and Wust, J.K. (1999). "A Unified Framework for Coupling Measurement in Object-Oriented Systems,"*IEEE Transactions on Software Engineering*, vol. 25, pp. 91-121.
[5] Abreu, F.B.e., (1995). "The MOOD Metrics Set," presented at ECOOP '95 Workshop on Metrics.
[6] Abreu, F. B. e.; Melo, W. (1996) "Evaluating the Impact of OO Design on Software Quality," presented at Third International Software Metrics Symposium, Berlin.
[7] Glasberg, D., et al. (2000). "Validating Object-Oriented Design Metrics on a Commercial Java Application," National Research Council 44146.

[8] Briand, L.; Emam, K.E.; Morasca, S. (1995). "Theoretical and Empirical Validation of Software Metrics.
[9] Briand, L., et al. (1981). "Measuring the Quality of Structured Designs," *Journal of Systems and Software*, vol. 2, pp. 113-120.
[10] Fowler, M., et al. (1999). *Refactoring: Improving the Design of Existing Code*. Reading, Massachusetts: Addison Wesley.
[11] Chidamber, S.R.; Kemerer, C. F. (1994). "A Metrics Suite for Object Oriented Design," *IEEE Transactions on Software Engineering*,Vol. 20.
[12] Churcher, N.I.; Shepperd, M.J. (1995). "Comments on 'A Metrics Suite for Object-Oriented Design'," *IEEE Transactions on Software Engineering*, Vol. 21, pp. 263-5.
[13] Pressman R.S. (2001). Software Engineering: A Practitioners' Approach, McGraw Hill International Edition, Fifth Edition.

[14] Blaha M.R.; Rumbaugh J.R. (2005). Object Oriented Modeling and Design with UML, Pearsons Publication, Second Edition