International Journal of Computer Applications (0975 – 8887)
Volume 181 – No. 21, October 2018

Cohesion as Software Design Decisive Measure: A Metric Approach

Poornima U. S.
Associate Professor
Department of Computer Science & Engineering
RR Institute of Technology
Bengaluru-India

Suma V.
Dean, Research and Industry Incubation Centre
Professor, Department of Information Science & Engineering
Dayanada Sagar College of Engineering,
Bengaluru-India

ABSTRACT

The requirements for developing software belongs to different domains in the current scenario keep evolving due to the instant changes and demand in the market. Hence, software design flexibility is a big challenge for the design architects to incorporate the changes as it occurs. The requirements gathered according to changes are grouped and implemented as modules which have their own responsibilities. Designing a module with complete functionality and integrating them is yet another challenge. Measuring such modules during design is therefore essential to make the final product qualitative. Further, it is worth to recall that quality of design is influenced by external quality attributes such as Cohesion, coupling, maintainability, scalability and so on. Further, cohesion concept is a qualitative indicator which decides the depth of design quality in any project. Therefore, this paper highlights on the impact of cohesion on design quality of a complex system and its measures to quantify the overall quality of software.

General Terms

Software Engineering, OOMD, Design Quality Metrics, Cohesion

Keywords

Software Quality, Software Solution-domain, Design Quality, Cohesion and Coupling

1. INTRODUCTION

Large-scale products developed under object oriented methodology have become inflexible in terms of design to accommodate the changes. The design quality of each module contributes to the quality of overall design. Cohesion reflects the binding of members within a module. Such binding are expected to be simple and single–task oriented under the control of the designer. Many cohesion types are identified which contributes to the quality of the whole module.

As the project grows, the number of modules increases and hence the integration and/or interdependency between them increase. Tracing out of such dependency along with coupling and keeping them under control is an area that needs to be focused. There is a continuous improvement needed to assess the quality of design to uphold the quality of final product. Such assessment is estimated by metrics which are the quantitative measures of different aspects of design. Thus, the product can be of good quality when the development team adapts simple, flexible and quality modules and respective metrics to assess it during development of a product [1].

The object oriented design methodology entitles bottom-up approach for product development. The data set is a prime requirement at the early stage of development process. Later phase includes the functionalities which are built upon data providing custom-specific services as a module. Software can be made more durable when the architecture representing such modules is flexible enough to accommodate the future changes, thereby upholding the scalability and maintainability of software.

Cohesion and Coupling (C&C) are generic concepts that represents good architecture of software which is not bound to any category of software. They are the measures of degree of connectivity of members within and outside of a module.

A software space domain includes set of modules interconnecting each other to provide service for the customers. Each module is framed of a class or set of classes which includes data and functions. Functions however operate on data to execute user requirements as services. Cohesion thus represents the relevance of existence of functions within a class or class within a module. Highly cohesive elements provide good services as well as increase the quality of design.

Additionally, cohesion is either on data or functions which represents the user services. Various cohesion types such as coincidental, logical, temporal, reflect on code cohesiveness rather than data. Such cohesion concentrates on grouping the related functions on logical similarity. However, communicational cohesion focuses on the binding of functions which access common data. Thus, in object oriented design, functional cohesion is grouping of classes in to packages based on functional similarity whereas, data cohesion is with respect to the wrapping of data with functions within a class.

2. REVIEW OF OOD QUALITY METRICS

Object-Oriented development methodology is a most popular methodology for multi-user, distributed and data-centric system. Class is a basic building block and vehicle for decomposition with operational attribute, methods and data attribute and data members. Therefore, success of the product depends on design quality of the software. Beside traditional metrics, many design quality metrics are proposed in the literature among which CK and MOOD metrics are popular in the literature.

2.1 CK (Chidamdert&Kemerer) Metrics

Object Model of the application domain has a class as basic building block and principles like Abstraction, Encapsulation, Inheritance and Polymorphism to make it complete.

Class: A basic unit of the solution framework with data and methods as attributes. The behaviour of the system is represented by methods coupling the classes through message passing.

A. *Weighted Methods per Class (WMC)* measures the time and effort required for developing and maintaining a class operational attribute; methods. A class Complexity is a cumulative sum of complexity of all its methods. The objective is to keep it low to uphold design quality.

WMC(C) =Σci(Mi) *i=1….n*

Where C is a class and M is a class method.

B. *Coupling Between Object classes (CBO)* measures the degree of interdependency between the classes. An object of a class can use the service or object of another class. The objective is to reduce cross coupling to increase the clarity of the solution

C. *Response For a Class (RFC)* measures response set of a class. When an object of a class sends a message, the methods executed inside and outside of a class are counted. The amount of effort in debugging, testing and maintenance is depending on response count.

|*RS*|= { M }U all i { Ri }

where { Ri } = set of methods called by method *i* and { M } = set of all methods in the class.

Inheritance Metrics: Size and complexity of the system is reduced by reusable components. Inheritance supports generalization and specialization concepts making the solution rich in terms of design.

A. Depth of Inheritance Tree (DIT) is a metric for measuring *vertical* growth of a class. Inheritance supports reusability. However, complexity is directly proportional to the distance between leaf and parent class. Hence, deeper tree structure is prone to higher complexity as it is difficult to access end class

behaviour.

B. Number Of Children (NOC) measures the *horizontal* growth of a class. The immediate subclasses in a hierarchy show the greater reusability. System functional quality is highly dependable on abstractness of the parent class. Hence, more effort is required in testing if tree grows in both directions.

Abstraction and Encapsulation: Identifying the properties for a problem has an impact on quality of the project. **Cohesive** methods make the architecture more sound, flexible and maintainable.

Lack of Cohesion in Methods (LCOM) measures the quality of a class in a solution domain. Cohesion refers the degree of interconnectivity between attributes of a class. A class is cohesive if it cannot be further divided in to subclasses. It measures the method behaviour and its relevance where it is defined. Pair of methods using data object proves the cohesiveness where as the methods not participating in data access makes it less cohesive. Consider C is a class and M1, M2,...Mn are its methods using set of class instances. Let I1={a,b,c,d}, I2={a,b,c} and I3={x,y,z} be the set of instances used by the methods M1,M2 and M3 respectively. If intersection of object set is non-empty then the methods using them is cohesive and their relevance in the class is proved. i.e. I1 I2={{a,b,c} means M1 and M2 are cohesive. But intersection of I1, I3 and I2, I3 is empty set. High count in LCOM shows less cohesiveness and class needs to be divided to subclasses. Several versions of the LCOM have been defined in the literature survey

2.2 High Level Design Metrics

Bansiya et al proposed a design-based class cohesion metric called Cohesion Among Methods in a Class (CAMC). The method-method interactions are considered in CAMC. This metric uses a parameter occurrence matrix with row for each method and a column for each data type that appears at least once as the type of a parameter in at least one method in the class. The value in row *i* and column *j* in the matrix equals 1 when *i*th method has a parameter of *j*th data type and equals 0 otherwise. In the matrix, the type of the class is always included in the parameter type list, and every method has an interaction with this data type, because every method implicitly has a self parameter. It indicates that columns is filled entirely with 1s. The CAMC metric is defined as the ratio of the total number of 1s in the matrix to the total size of the matrix. Counsell et al. suggest omitting the type of the class from the parameter occurrence matrix and calculating CAMC.

Counsell et al. propose design-based class cohesion metric namely, Normalized Hamming Distance (NHD) based on hamming distanc. In this metric, only the method-method interactions are considered. The metric uses the same parameter occurrence matrix used by CAMC metric (the type of the class is not considered). The metric calculates the average of the parameter agreement between each pair of methods. The parameter agreement between a pair of methods is defined as the number of places in which the parameter occurrence vectors of the two methods are equal. Formally, the metric is defined as follows:

3. LITERATURE SURVEY

Authors of [2] proposed 6 metrics on classes to explore the relationship between class attributes. The metrics are both dynamic and static in nature to measure the cohesion.

Authors of [3] proposed flattening functions in the hierachy among superclass and subclass. He considered both attributes and methods with and without conflicting names to build attribute and methods set in the subclass. He argued that hierachy not only provides reusabilty but also increases the complexity. He also argued that complexity rises from before to after flatting the classes. The size, cohesion and coupling metrics were measured to justify flattening impact on complexity.

Author of [4] explain how and when class flattening happens in Java. He further concludes that there is a need to study the impact of flattening when using internal quality attributes to indicate external quality attributes.

Authors of [5] proposed an inheritance metric based on UML diagram at design phase. The metric is empirically validated against Weyuker axioms and proved same as DIT of CK metric.

Authors of [6] proposed an inheritance metric DITC based on number of attributes and methods at each level of hierarchy. The metric is theoretically validated and further indicates effect on the development time (DEV).

Authors of [7] proposed two inheritance metrics, ICC (Inheritance Complexity of a Class) and ICT (Inheritance Complexity of a Tree). He further proved with many cases that interaction increases the complexity(W9 property).

Authors of [8] discussed the difference between inheritance and interface in C# programming. They calculated cohesion and coupling values of different projects and concluded that interface has less coupling value than inheritance and more reusable.

Authors of [9] proposed a metric NOPD (Number of Polymorphic Dispatches) for inheritance hierarchy. He stated that NOPD is useful in designing the test cases for inheritance structure. The paper concludes that NOPD presents number of test cases for the current hierarchical design of the project.

Authors of [10] had done the empirical study on evaluating the depth of inheritance on maintainability of Object Oriented Software. They designed and conducted experiments to prove influence of inheritance on maintainability.

Authors of [11] measured the design complexity using all inheritance metrics. They also measured the complexity with class interfaces and proved that interface is better than inheritance.

4. CLASS INHERITANCE METRICS

Class can be considered as set of attributes and methods. The axiom of pairing is a basis for visualizing a class as *a set* at static design level. The singleton axiom proposes a set as collection of elements. However, pairing axiom exhibits the property that a set can be a *collection of exactly two unordered pairs of sets.*

S = {A, B}

where A ≠ B. However, a class at static design level can also be visualized as a set since it contains collection of data and functions which are logically interconnected.

C= {D, M}

where D={ d1,d2,d3,d4} and M={ f1,f2,f3} are logically related subsets representing C as class, D as class data attributes and M as class methods.

The strength of a class depends on the degree of relatedness among its members, attributes and methods. Many cohesion metrics such as Lack of Cohesion among Methods (LCOM, LCOM1, LCOM2, LCOM3, LCOM4, LCOM5), Tight Class Cohesion (TCC), Loose Class Cohesion (LCC), Degree of Cohesion-Direct (DCD), Degree of Cohesion Indirect (DCI), Sensitive Class Cohesion Metric (SCOM) etc, measures the connectivity between members. Consider C as a class and M1, M2,...Mn are its methods with set of class instances. Let I1={a,b,c,d}, I2={a,b,c} and I3={x,y,z} are set of instances initiated by the methods M1,M2 and M3 respectively. If intersection of object set is non-empty then the methods using them is cohesive and their relevance in the class is proved. i.e. I1 ∧ I2={a,b,c} which infers that M1 and M2 are cohesive. But intersection of I1, I3 and I2, I3 is empty set. High count in LCOM hence shows less cohesiveness and class need to be divided to subclasses. Similarly, other cohesive metrics include variations of LCOM by considering the different variations of methods and attributes. When applying set relations to class, inheritance holds transitive property. In the hierarchy of three classes, the super class c1 inherits to subclass c2 and c2 to c3. Therefore,

(c1, c2) ∈ R and (c2, c3) ∈ R -> (c1, c3) ∈ R

Where, relation R stands for inheritance. Thus, transitive relativity applies to inheritance where class c3 inherits properties of c1 through c2.

In recent years, research is more focused on finding the complexity of the inheritance tree since it directly impacts the class cohesion. There are many inheritance metrics proposed which measures the depth of reusability in an inheritance ladder as shown in Table 1.

Table 1. Class Inheritance Metrics

Sl No	Metric Name	Description	Proposed by
1	DIT	Maximum length from the node to root node	C& K 1994
2	NOC	Number of immediate subclasses of a root	C & K 1994
3	DITC	Sum of attributes, methods at a class level by considering all visibility modes L DITC(Ci)=Σ*LEV i*1* i=1 LEVi = Attribute (Ci) + Method (Ci)	Kumar rajnish Vandana Bhattacherjee 2006
4	NOPD	Number of Polymorphic Dispatches	Naveen Sharma, Padmaja Joshi, and Rushikesh K. Joshi 2006
5	DIC	Degree of Inheritance of a Class DIC=Number of inherited Attributes/Methods x (4 - level) if level <= 3 DlC = Number of inherited Attributes/Methods x (level - 3) if level >= 4	Gagandeep Makkar\\ Jitender Kumar Chhabra2 and Rama Krishna Challa3 2012

5. DIFFERENT PROPERTIES FOR VALIDATING METRICS

***LIONEL BRIAND* et al, Properties**

Lionel Briand, et al had defined interesting strategies to define the metrics at High level Design. They considered all possible interactions of code segments and proposed several corresponding metrics. They also proposed 9 properties using which the metrics for cohesion and coupling are validated.

1 Normalization

Given a software part *sp*, the metric *cohesion (sp)* belongs to a specified interval *[0,Max]*, and *cohesion(sp)* = 0 if and only if *CI(sp)* is empty, and *cohesion(sp) = Max* if and only if *CI(sp)* includes all possible cohesive interactions.

Thus, class cohesiveness is expected either *max or 0* to get the design quality.

2. Monotonicity

Let sp1 be the software part and CI(sp1) is the set of cohesive interactions. Let sp2 is modified sp1 with one more interaction. Then *cohesion(sp2)>=cohesion(sp1).*
This property illustrates that adding interaction will not decrease the cohesion.

3. Cohesive modules
Let sp1 be the software part and *m1*, *m2* are two modules belong to sp1. Let sp2 is a new software part with new module m=m1+m2. If no cohesive interactions exist between the declarations belonging to *m1* and *m2* when they are grouped in *m*, then *cohesion (sp1)>=cohesion(sp2).*
Thus, the cohesion of the merged class is less than its individual classes.
When defining metrics for cohesion, either during high level or code level, the metrics are validated for the better design quality.

6. PROPOSED WORK AND CONCLUSION

Quality is an expected property of software. Different activities of software development can imbibe the quality, one of which phase is design during software development. everal researchers had proposed the metrics for design which directly influences both internal quality and external factors of software such as maintainability, scalability, testability etc,.

Since cohesion is one of the major design quality decisive factor, it must be measured. When the code is reused, the code quality must be tested. The subclass inherits its specialized features. Several inheritance metrics are defined, but its relation with measuring the cohesion is yet the research concept. This work may give insight to maximum length of hierarchy or reusability encouraged during design phase itself. Refactoring of classes can be done based on both inheritance and cohesion values.

7. REFERENCES

[1] Jehad Al Dalal, “Measuring the Discriminative Power of Object-Oriented Class Cohesion Metrics “,IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 37, NO. 6, NOVEMBER/DECEMBER 2011.

[2] Shyam R. Chidamber and Chris F. Kemerer, “A Metrics Suite for Object Oriented Design”, IEEE TRANSACTION ON SOFTWARE ENGINEERING, VOL 20,No 6, JUNE 1994.

[3] Dirk Beyer, Claus Lewerentz, and Frank Simon. “Impact of Inheritance on Metrics for Size, Coupling, and Cohesion in Object-Oriented Systems”, Software Systems Engineering Research Group ,Technical University Cottbus, Germany (db|cl|simon)@informatik.tu-cottbus.de.

[4] Jehad Al Dallal, “how and when to flatten java classes?”, International Journal of Computer Science, Engineering and Information Technology (IJCSEIT), Vol. 4, No.2, April 2014.

[5] Gagandeep Makkar\ Jitender Kumar Chhabra2 and Rama Krishna Challa3. “Object Oriented Inheritance MetricReusability perspective” . 20 1 2 International Conference on Computing, Electronics and Electrical Technologies [ICCEET].

[6] Kumar rajnish, vandana bhattacherjee, “Class Inheritance Metrics-An Analytical and Empirical Approach”, 1Department of Computer Science & Engineering, Birla Institute of Technology, Ranchi-01, India.

[7] Sandip mal, Rajnish Kumar, “Applicability of Weyuker’s Property 9 to Inheritance Metric” *International Journal of Computer Applications (0975 – 8887) Volume 66– No.12, March 2013.*

[8] Maya Yadav, Jasvinder pal Singh, Pradeep Baniya, “ Complexity Identification of Inheritance and Interface based on Cohesion and Coupling Metrics to Increase Reusability “, *International Journal of Computer Applications (0975 – 8887) Volume 64– No.8, February 2013.*

[9] Naveen Sharma, Padmaja Joshi, and Rushikesh K. Joshi, “Applicability of Weyuker’s Property 9 to Object Oriented Metrics”, IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 32, NO. 3, MARCH 2006.

[10] John Dely, Andrew Brooks, et al, “ An Empirical Study Evaluating Depth of Inheritance on the Maintainability of Object-Oriented Software, Empirical Studies of Programmers: Sixth Workshop. Intellect, pp. 39-58. ISBN 9781567502626 **.**

[11] Varsha Mishra, Shweta Yadav, “ Quality evaluation of factors affecting the reusability of object oriented class inheritance and interface”, International Journal of Research in Engineering Technology and Management ISSN 2347 – 7539.