ISSN: 2456 – 3137
International Journal of Computational Research and Development
Impact Factor 4.775, Special Issue, January - 2017
International Conference on Smart Approaches in Computer Science Research Arena
On 5th January 2017 Organized By
Department of Computer Science, Sri Sarada College for Women (Autonomous), Salem, Tamilnadu

AN OVERVIEW AND COMPARATIVE STUDY ON DIFFERENT OBJECT ORIENTED DESIGN COMPLEXITY METRICS

Dr. M. Latha
Associate Professor of Computer Science, Sri Sarada College for Women, Salem, Tamilnadu

Cite This Article: Dr. M. Latha, "An Overview and Comparative Study on Different Object Oriented Design Complexity Metrics", International Journal of Computational Research and Development, Special Issue, January, Page Number 16-19, 2017.

Abstract:

Software metrics are quantitative estimates for software product attributes which guide in taking managerial and technical decisions. Software metrics measure different aspects of software complexity and therefore play an important role in analysing and improving software quality. Object-oriented modelling and design is a way of thinking about problems using models organized around real world concepts. Various object oriented metrics with other metrics like traditional metrics, orthogonal metrics and CK metrics were discussed. This comparative study on different object oriented metrics shows a direct positive correlation between the degree of object oriented constructs and the level of quality for each software application. Object oriented design is becoming more popular in software development environment and object oriented design metrics is an essential part of software environment. This study focus on a set of object oriented metrics that can be used to measure the quality of an object oriented design. The metrics for object oriented design focus on measurements that are applied to the class and design characteristics. These measurements permit designers to access the software early in process, making changes that will reduce complexity and improve the continuing capability of the design.

Key Words: Object Oriented Metrics, Traditional Metrics & Orthogonal Metrics

1. Introduction:

Object-oriented modelling and design is a way of thinking about problems using models organized around real-world concepts. The fundamental construct is the object, which combines both data structure and behaviour in a single entity. There are three fundamental characteristics required for an object-oriented approach: encapsulation, polymorphism, and inheritance. Encapsulation is not unique to the object-oriented paradigm; however polymorphism and inheritance are two aspects unique to the object-oriented approach. These three aspects of the object-oriented paradigm are described below.

Encapsulation:

Encapsulation is a method to wrapping the external aspects of an object, which are accessible to other objects from the internal implementation details of the object that are hidden from other objects. Encapsulation prevents a program from becoming so interdependent that a small change has massive effects. For example, the implementation of an object can be changed without affecting the application that uses it. One may want to change the implementation of an object to improve performance, fix a bug, consolidate code, or for porting.

Polymorphism:

Polymorphism means having the ability to take several forms. For object-oriented systems, polymorphism allows the implementation of a given operation to be dependent on the object that contains the operation. For example, In an organization a single employee may play many fertilise roles in an organization, so it shows the ability to take several forms.

Inheritance:

Acquiring the properties of other objects, it allows programmers to define objects incrementally by reusing previously defined objects as the basis for new objects. For example, sharing of properties from parents to kids. In simple words we say that, Inheritance is a reuse mechanism.

2. Object Oriented Metrics:

- ✓ Weight Method per Class (WMC)
- ✓ Response for Class (RFC)
- ✓ Lack of Cohesion of Methods (LCOM)
- ✓ Coupling between Object Classes (CBO)
- ✓ Depth of Inheritance Tree (DIT)
- ✓ Number of Children (NOC)

2.1 Weight Method per Class (WMC): This metric is used to measure the understand ability, reusability and maintainability.

- ✓ A class is a template from which objects can be created. Classes with large number of methods are likely to more application specific, limiting the possibility of reuse.
- ✓ This set of objects shares a common structure and a common behavior manifested by the set of methods.
- ✓ The WMC is a count of the methods implemented within a class or the sum of the complexities of the methods. But the second measurement is more difficult to implement because not all methods are accessible within the class hierarchy because of inheritance.
- ✓ The larger the number of methods in a class is the greater the impact may be on children, since children inherit all of the methods defined in a class.

2.2 Response for Class (RFC): A message is a request that an object makes to another object to perform an operation. The operation executed as a result of receiving a message is called a method.

ISSN: 2456 – 3137
International Journal of Computational Research and Development
Impact Factor 4.775, Special Issue, January - 2017
International Conference on Smart Approaches in Computer Science Research Arena
On 5th January 2017 Organized By
Department of Computer Science, Sri Sarada College for Women (Autonomous), Salem, Tamilnadu

- ✓ The RFC is the total number of all methods within a set that can be invoked in response to message sent to an object. This includes all methods accessible within the class hierarchy.
- ✓ This metrics is used to check the class complexity. If the number of method is larger that can be invoked from class through message than the complexity of the class is increase

2.3 Lack of Cohesion of Methods (LCOM): Cohesion is the degree to which methods within a class are related to one another and work together to provide well bounded behavior.

- ✓ LCOM uses variable or attributes to measure the degree of similarity between methods.
- ✓ We can measure the cohesion for each data field in a class; calculate the percentage of methods that use the data field.
- ✓ Average the percentage, then subtract from 100 percent. Lower percentage indicates greater data and method cohesion within the class.
- ✓ High cohesion indicates good class subdivision while a lack of cohesion increases the complexity.

2.4 Coupling between Object Classes (CBO):

- ✓ Coupling is a measure of strength of association established by a connection from one entity to another.
- ✓ Classes are couple in three ways. One is, when a message is passed between objects, the object are said to be coupled. Second one is, the classes are coupled when methods declared in one class use methods or attributes of the other classes. Third on is, inheritance introduced significant tight coupling between super class and subclass.
- ✓ CBO is a count of the number of other classes to which a class is coupled. It is measured by counting the number of distinct non inheritance related class hierarchy on which a class depends.
- ✓ Excessive coupling is detrimental to modular design and prevent reuse. If the number of couple is larger in software than the sensitivity to changes in other in other parts of design.

2.5 Depth of Inheritance Tree (DIT):

- ✓ Inheritance is a type of relationship among classes that enables programmers to reuse previously defined object objects, including variables & operators.
- ✓ Inheritance decrease the complexity by reducing the number of operations and operators, but this abstraction of objects can make maintenance and design more difficult.
- ✓ Depth of class within the inheritance hierarchy is the maximum length from the class node to the root of the tree, measured by the number of ancestor classes.
- ✓ The deeper a class within the hierarchy, the greater the number of methods and is likely to inherit, making it more complex to predict its behavior.
- ✓ A support metric for DIT is the number of methods inherited.

2.6 Number of Children (NOC):

- ✓ The number of children is the number of immediate subclasses subordinates to class in the hierarchy.
- ✓ The greater the number of children, the greater the parent abstraction.
- ✓ The greater the number of children, greater the reusability, since the inheritance is a form of reuse.
- ✓ If the number of children in class is larger than it require more testing time for testing the methods of that class.

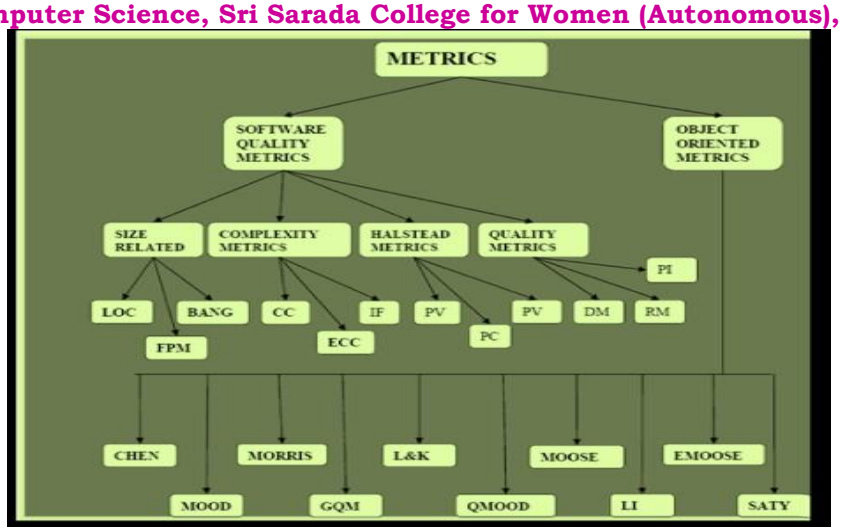
3. Traditional Metrics:

Traditional metrics have been applied to the measurement of software complexity of structured systems since1976 [MCC76]. This subsection presents the McCabe Cyclomatic Complexity metric along with two other popular traditional software design metrics, Source Lines of Code and Comment Percentage. McCabe Cyclomatic Complexity (CC): Cyclomatic complexity is a measure of a module control flow complexity based on graph theory [MCC99]. Cyclomatic complexity of a module uses control structures to create a control flow matrix, which in turn is used to generate a connected graph. The graph represents the control paths through the module. The complexity of the graph is the complexity of the module [MCC76], [MCC99]. Fundamentally, the CC of a module is roughly equivalent to the number of decision points and is a measure of the minimum number of test cases that would be required to cover all execution paths. A high cyclomatic complexity indicates that the code may be of low quality and difficult to test and maintain.Source Lines of Code (SLOC): The SLOC metric measures the number of physical lines of active code, that is, no blank or commented lines code [LOR94]. Counting the SLOC is one of the earliest and easiest approaches to measuring complexity. It is also the most criticized approach [TEG]. In general the higher the SLOC in a module the less understandable and maintainable the module is.Comment Percentage (CP): The CP metric is defined as the number of commented lines of code divided by the number of non-blank lines of code.

4. Orthogonal Object-Oriented Metrics:

Orthogonal: Orthogonality is a measure of intrinsically different characteristics of the code, therefore anycorrelation among the measured values is due to relationships among the target modules and not due to any relationships among the actual metrics themselves. For example, lines of code and number of comments are said to be non-orthogonal since adding comments simultaneously increases lines of code. However, source lines of code and number of comments are said to be orthogonal since source lines of code can be increased without any changes in the comment count.

ISSN: 2456 – 3137
International Journal of Computational Research and Development
Impact Factor 4.775, Special Issue, January - 2017
International Conference on Smart Approaches in Computer Science Research Arena
On 5th January 2017 Organized By
Department of Computer Science, Sri Sarada College for Women (Autonomous), Salem, Tamilnadu



Orthogonal Object-Oriented (OOO): Two object-oriented metrics are said to be Orthogonal Object-Oriented metrics if they are orthogonal.

5. CK Metrics:

CK metrics were designed:

- ✓ To measure unique aspects of the OO approach.
- ✓ To measure complexity of the design.
- ✓ To improve the development of the software

Analysing CK metrics through the identification of *outlying* values (extreme deviations), which may be a signal of:

- ✓ high complexity and/or
- ✓ possible design violations

6. Literature Review:

The evolution categories proposed by Lanza were first described in his study on using evolution matrices tovisualize the evolution of object-oriented software systems (Lanza 2001). Rows in evolution matrices represent classes, whereas columns denote versions of the target system. Each cell encodes two metrics at a time: its height scales to the number of instance variables (NIV) of the class denoted by the containing row, and its width is proportional to the number of methods (NOM) in the same class. Lanza evaluates evolution Software Qual J matrices using two Smalltalk systems, in which supernovas, white dwarfs, pulsars, stagnant, and dayfly classes are visually identified in terms of NIV and NOM. The evaluated systems, however, are not representative of industrial-strength object-oriented systems. Lanza and Ducasse (2003) propose the use of polymeric views, which contrary to Lanza's evolution matrices, can encode up to five metrics measurements. A polymeric view is a graph representing a given relationship between source code entities (e.g.,classes), where nodes encode metric measurements by means of colours, position, and size. Colours are in gray scale, with white standing as the least value and black the maximum metric measurement. Positions are (x and y) coordinates, and node size encodes two measurements by its size and Israeli and Feitelson perform a larger study, with the analysis of 810 Linux kernel releases over 14 years (Israeli and Feitelson 2010). Their analysis suggests that the kernel agrees with Lehman's Law of Software Evolution. In particular, the authors report strong evidence toward continuing growth and change laws, but anecdotal evidence of the self-regulation and feedback system laws. Gonzalez-Barahona et al. (2009) study the evolution of the Debian Linux distribution, opposed to studies focusing on the kernel alone, and point out that the package mean size in Debian is often constant across stable releases. However, the number of packages and the LOC size of the distribution (the sum of all LOC in each source file) double at each release. They also find that 7 % of packages from version 2.0 are still present in version 4.0, and that around 18 % of such packages remained unchanged (something we would characterize as a stagnant behaviour).Herraiz and Hassan (2010) investigate the correlation between LOC and other software metrics. In particular, the authors analyze how LOC/SLOC measurement of C files in Arch Linux packages correlates with McCabe's control flow complexity and Hal- stead's metrics. Overall, they find a high correlation (C 84 %) between size and Hal- stead's metrics, but an average correlation (60 %) between SLOC and McCabe'scyclomatic complexity for non-header files. Finally, a low correlation between McCabe's complexity and SLOC/LOC is present. As discussed by the authors, this is expected, as header files do not contain control flow information. Similar results are reported in Herraiz et al. (2007), but with FreeBSD as a subject of analysis. Altogether, these works measure correlations between metric values taken from a single version of the target systems, which are restricted to the domain of operating systems.

7. Comparative Study between Metrics:

A Simple and easy way to use minimal set of Orthogonal Object-Oriented metrics in the form of an equation and one standalone metric, which can be used to evaluate software quality, using the CK suite of object-oriented metrics as a superset were constructed. This was accomplished by observing that the correlation of the metrics contained in the CK metrics suite increased as the quality of code decreased. This motivated a relationship between some of the metrics in the CK suite of object-oriented metrics. The reduced metrics suite was validated using three industrial strength software systems. By comparing the results

ISSN: 2456 – 3137
International Journal of Computational Research and Development
Impact Factor 4.775, Special Issue, January - 2017
International Conference on Smart Approaches in Computer Science Research Arena
On 5th January 2017 Organized By
Department of Computer Science, Sri Sarada College for Women (Autonomous), Salem, Tamilnadu

obtained from the reduced metrics set approach with the results obtained from a full-scale code analysis conducted using the entire CK object-oriented metrics suite together with traditional metrics. The reduced metrics set approach was able to classify the software systems with respect to the level of code quality. Both the reduced metrics set approach and the full metrics set (CK metrics suite and traditional) approach resulted in the same software quality system classification. System A was high quality software, System B was medium, and System C low. Table 1 summarizes this descriptive information along with other information for each System. The last two rows in the table were obtained from the SATC full-scale code analysis of these systems. The table shows a direct positive correlation between the degree of object-oriented constructs and the level of quality for each software application.

Table 1: comparison between applications used to validate reduced orthogonal and ck metrics set.

System	A	B	C
Lines of code used	148K	295K	650K
Number of classes Language	52K	1900	2517
Language type of application	Java	Java	C++
Code Construct	Orthogonal and CK metrics	Excellent Object Oriented Metrics	Traditional Object Oriented Metrics
Quality	High	Medium	Low

8. Conclusion and Future Enhancement:

A metrics program that is based on the goals of an organization will help communicate, measure progress towards, and eventually attain those goals. In this paper, different type of object oriented design complexity metrics were discussed and compared, which are used during the software development. The approach is more applicable to identify low quality code than high, due to specific theoretical concepts employed to develop the approach. However, this is a mammoth step in the right direction in reducing the turnaround time it takes to perform a code analysis on industrial strength software. Future research should be conducted with the aim of developing more appropriate reduced metrics set models for identifying high quality code and how this reduced object-oriented metrics set approach can be integrated into the software development lifecycle.

9. References:

1. Chidamber, Shyam and Kemerer, Chris, “A metrics Suite for Object Oriented Design”, IEEE Transactions on Software Engineering, June, 1994, pp. 476-492.
2. Lorenz, Mark and Kidd, Jeff, Object Oriented Software metrics, Prentice Hall Publishing, 1994.
3. Victor R. Basili, Lionel Briand and Walcelio L. Melo “A validation of object-oriented design metrics as quality indicators” Technical report, Uni. of Maryland, Deptt. of computer science, MD, USA. April 1995.
4. “The Role of Object Oriented metrics” from archive.eiffel.com/ doc/ manuals/ technology.
5. Bohem, B.W, Brown,j.R. and Lipow, M, “Quantitve evaluation of software quality” proceedings of the second international conference on software engineering, 1976.
6. Norman E.Fenton, shari Lawerance Pfleeger. Software metrics Arigorous and practical approach. Second edition.PWS Publishing Company. 20 park plaza Boston.
7. Dromy. R.G, “cornering the chimera”, IEEE software, 31(1), 33-34, 1996.
8. Norman E. Fenton and Shari Lawrence Pfleeger. Software Metrics: A Rigorous and Practical Approach. PWS Publishing Company, 1997.