International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Index Copernicus Value (2015): 78.96 | Impact Factor (2015): 6.391

Improvement in the Quality of the Object-Oriented System

Vijay Kumar Tiwari

Department of CSE, Kamla Nehru Institute of Technology, Sultanpur-228118, U.P., India

Abstract: ***This research seeks to describe concepts and techniques to improve the quality of the object-oriented System. Object-oriented design is a popular concept in today's system development. Many metrics relating to product quality have proved their value for system maintenance and modification. The paper introduces the use of three traditional metrics and presents six additional metrics specifically for object-oriented systems. The object-oriented metrics criteria selected are used to evaluate the system attributes. A brief description of the object-oriented structure is given. Each metric is then described, interpretation guidelines given, and the applicable quality attributes listed. Future work will be to define criteria for the metrics. Acceptable ranges for each metric will have to developed based on the effect of the metric on desirable system qualities.***

Keywords: Object-oriented design, object-oriented system, system metric, system.

1. Introduction

With today's system development, object-oriented design is a popular concept. It has proved its value for system that maintained and modified. Object-oriented system development requires a different approach from more traditional functional decomposition and data flow development methods [1]. It uses the system metrics to evaluate the Quality of the object-oriented system. With object-oriented analysis and design methodologies gaining popularity, it is time to start investigating object-oriented metrics with respect to system quality.

1) Complexity: The systems can be used more effectively to decrease the architectural complexity.
2) Efficiency: The systems are efficiently designed.
3) Reusability: The design quality supports possible reuse.
4) Testability/Maintainability: The systems support ease of testing and changes.
5) Understandability: The design increases the psychological complexity.

The former approach to identify a set of object-oriented metrics was to focus on the primary, critical constructs of object-oriented design and to select metrics that apply to those areas [4]. The metrics are supported by most literature and some object-oriented tools. The metrics evaluate the object-oriented concepts: methods, classes, coupling, and inheritance. The metrics focus on internal object structure, external measures of the interactions among entities, measures of the efficiency of an algorithm and the use of machine resources, as well as psychological measures that affect the ability of a programmer to create, comprehend, modify and maintain system [5].

The paper supports the use of three traditional metrics and presents six additional metrics specifically for object-oriented systems. Some researchers and practitioners contend traditional metrics are inappropriate for object-oriented systems. There are valid reasons for applying traditional metrics, however, if it can be done. The traditional metrics have been widely used, they are well understood by researchers and practitioners, and their relationships to system quality attributes have been validated [6].

2. Traditional Metrics for Object-Oriented System

Metrics or the Evaluation Criteria

This paper takes the radical approach to completely replace the While metrics for the traditional functional decomposition and data analysis design approach measure the design structure and data structure independently, object-oriented metrics must be able to focus on the combination of function and data as an integrated object [2]. The evaluation of the utility of a metric as a quantitative measure of system quality was based on the measurement of a system quality attribute. The object-oriented metric criteria are to be used to evaluate the following attributes [3]:

In an object-oriented system, traditional metrics are generally applied to the methods that comprise the operations of a class. A method is a component of an object that operates on data in response to a message and is defined as part of the declaration of a class. Three traditional metrics are discussed here: cycloramic complexity, size (line counts) and comment percentage [7].

Metric 1: Cycloramic Complexity (CC)

Cycloramic complexity is used to evaluate the complexity of an algorithm in a method. A method with a low cycloramic complexity is generally better, although it may mean that decisions are deferred through message passing, not that the method is not complex. Cycloramic complexity cannot be used to measure the complexity of a class because of inheritance, but the cycloramic complexity of individual methods can be combined with other measures to evaluate the complexity of the class. In general, the cycloramic complexity for a method should be below ten, indicating decisions are deferred through message passing. Although this metric is specifically applicable to the evaluation of quality attribute Complexity, it also is related to all of the other attributes [8].

Metric 2: Size
Size of a method is used to evaluate the ease of understandability of the code by developers and maintainers. Size can be measured in a variety of ways. These include counting all physical lines of code, the number of statements and the number of blank lines. Thresholds for evaluating the size measures vary depending on the coding language used and the complexity of the method. However, since size affects ease of understanding, routines of large size will always pose a higher risk in the attributes of Understandability, Reusability, and Maintainability [9].

Metric 3: Comment Percentage
The line counts done to compute the Size metric can be expanded to include a count of the number of comments, both on-line (with code) and stand-alone. The comment percentage is calculated by the total number of comments divided by the total lines of code less the number of blank lines. The SATC has found a comment percentage of about 30%. Since comments assist developers and maintainers, this metric is used to evaluate the attributes of Understandability, Reusability, and Maintainability [10].

3. Specific Metrics for Object-Oriented System

As we know, many different metrics have been proposed for object-oriented systems. The object-oriented metrics that were chosen by the SATC measure principle structures that, if improperly designed, negatively affect the design and code quality attributes. The selected object-oriented metrics are primarily applied to the concepts of classes, coupling, and inheritance. For some of the object-oriented metrics discussed here, multiple definitions are given, since researchers and practitioners have not reached a common definition or counting methodology. In some cases, the counting method for a metric is determined by the system analysis package being used to collect the metrics [11].

A. Class
A class is a template from which objects can be created. This set of objects share a common structure and a common behaviour manifested by the set of methods. Three class metrics described here measure the complexity of a class using the class's methods, messages and cohesion.

1) Method
A method is an operation upon an object and is defined in the class declaration.

Metric 4: Weighted Methods per Class (WMC)
The WMC is a count of the methods implemented within a class or the sum of the complexities of the methods (method complexity is measured by cycloramic complexity). The second measurement is difficult to implement since not all methods are accessible 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 since children inherit all of 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 metric measures understandability, maintainability, and reusability.

2) Message
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. The next metric looks at methods and messages within a class.

Metric 5: Response for a Class (RFC)
The RFC is the carnality of the set of all methods that can be invoked in response to a message to an object of the class or by some method in the class. This includes all methods accessible within the class hierarchy. This metric looks at the combination of the complexity of a class through the number of methods and the amount of communication with other classes. The larger the number of methods that can be invoked from a class through messages, the greater the complexity of the class. If a large number of methods can be invoked in response to a message, the testing and debugging of the class becomes complicated since it requires a greater level of understanding on the part of the tester. A worst case value for possible responses will assist in the appropriate allocation of testing time. This metric evaluates Understandability, Maintainability, and Testability [12].

3) Cohesion
Cohesion is the degree to which methods within a class are related to one another and work together to provide well-bounded behaviour. Effective object-oriented designs maximize cohesion since it promotes encapsulation. The third class metrics investigates cohesion.

Metric 6: Lack of Cohesion of Methods (LCOM)
LCOM measures the degree of similarity of methods by data input variables or attributes (structural properties of classes). Any measure of separateness of methods helps identify flaws in the design of classes. There are at least two different ways of measuring cohesion:

1) Calculate for each data field in a class what percentage of the methods use that data field. Average the percentages then subtract from 100%. Lower percentages mean greater cohesion of data and methods in the class.
2) Methods 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 metric evaluates Efficiency and Reusability.

4) Coupling
Coupling is a measure of the strength of association established by a connection from one entity to another. Classes (objects) are coupled three ways:

1) When a message is passed between objects, the objects are said to be coupled.

2) Classes are coupled when methods declared in one class use methods or attributes of the other classes.
3) Inheritance introduces significant tight coupling between superclasses and their subclasses. Since good object-oriented design requires a balance between coupling and inheritance, coupling measures focus on non-inheritance coupling.

Metric 7: Coupling Between Object Classes (CBO)
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 hierarchies on which a class depends. Excessive coupling is detrimental to modular design and prevents reuse. The more independent a class is, the easier it is reuse in another application. The larger the number of couples, the higher the sensitivity to changes in other parts of the design and therefore maintenance is more difficult. Strong coupling complicates a system since a module is harder to understand, change or correct by itself if it is interrelated with other modules. Complexity can be reduced by designing systems with the weakest possible coupling between modules. This improves modularity and promotes encapsulation. CBO evaluates Efficiency and Reusability.

B. Inheritance
Another design abstraction in object-oriented systems is the use of inheritance. Inheritance is a type of relationship among classes that enables programmers to reuse previously defined objects including variables and operators. Inheritance decreases complexity by reducing the number of operations and operators, but this abstraction of objects can make maintenance and design difficult. The two metrics used to measure the amount of inheritance are the depth and breadth of the inheritance hierarchy [13].

Metric 8: Depth of Inheritance Tree (DIT)
The depth of a class within the inheritance hierarchy is the maximum length from the class node to the root of the tree and is measured by the number of ancestor classes. The deeper a class is within the hierarchy, the greater the number methods it is likely to inherit making it more complex to predict its behavior. Deeper trees constitute greater design complexity, since more methods and classes are involved, but the greater the potential for reuse of inherited methods. A support metric for DIT is the number of methods inherited (NMI). This metric primarily evaluates Efficiency and Reuse but also relates to Understandability and Testability.

Metric 9: Number of Children (NOC)
The number of children is the number of immediate subclasses subordinate to a class in the hierarchy. It is an indicator of the potential influence a class can have on the design and on the system. The greater the number of children, the greater the likelihood of improper abstraction of the parent and may be a case of misuse of subclassing. But the greater the number of children, the greater the reusability since inheritance is a form of reuse. If a class has a large number of children, it may require more testing of the methods of that class, thus increase the testing time. NOC, therefore, primarily evaluates Efficiency, Reusability, and Testability [14].

4. Conclusion

Product Quality for code and design has five attributes. These are Efficiency, Complexity, Understandability, Reusability, and Testability/Maintainability. The SATC has proposed nine metrics for object-oriented systems. They cover the key concepts for object-oriented designs: methods, classes (cohesion), coupling, and inheritance. For each metric, threshold values can be adopted, depending on the applicable quality attributes and the application objectives. Future work will be to define criteria for the metrics. That is, acceptable ranges for each metric will have to developed, based on the effect of the metric on desirable software qualities.

References

[1] R. Hudli, C. Hoskins, and A. Hudli, "Software Metrics for Object-oriented Designs," *IEEE Trans. Electron Devices*, vol. 2, pp. 314-319, Jan. 1993.
[2] Y. Lee, B. Liang, and F. Wang, "*Some Complexity Metrics for Object-Oriented Programs Based on Information Flow*," in *Proc. CompEuro*, March, 1993, pp. 302-310.
[3] M. Lorenz and J. Kidd, *Object-Oriented Software Metrics*, Prentice *Hall Publishing*, 1994, ch. 2, pp. 425-427.
[4] D. Tegarden, S. Sheetz, and D. Monarchi, "Effectiveness of Traditional Software Metrics for Object-Oriented Systems," in *Proc. 25th Hawaii International Conference on System Sciences*, January, 1992, pp. 359-368.
[5] H.-Y. Song, F. Hao, M. Kodialam, and T. V. Lakshman, "IPv6 lookups using Distributed and Load Balanced Bloom Filter for 100Gbps Core Router Line Cards," *INFOCOM*, 2009, pp. 425-447.
[6] G. Millar, E. Panaousis, and C. Politis, "Robust: Reliable overlay based utilisation of services and topology for emergency manets," in *Proc. IEEE Future Network and MobileSummit, Florence*, Italy, June 2010,
pp. 75-82.
[7] Z.-Q. Xia, Z.-G. Chen, and X.-H. Deng, "An Enforceable Incentive Scheme in Wireless Multi-path Inter-session Network Coding Game," *Journal of Networks*, ISSN 1796-2056, vol. 7, issue 2, 2012, pp. 351-355.
[8] I. Christian, G. Lorenza, and A. Sateesh, "Distributed Multiple Access and Flow Control for Wireless Network Coding," *Vehicular Technology Conference (VTC 2010-Spring)*, 16-19 May 2010, pp. 1–6, Location: Taipei.
[9] V. Reddy, S. Shakkottai, A. Sprintson, and N. Gautam. "Multipath wireless network coding: a population game perspective," in *Proc. IEEE INFOCOM 2010*, San Diego, March 2010, pp. 1-9.
[10] T. T. Chen and S. Zhong. "INPAC: An enforceable incentive scheme for wireless networks using network coding," *IEEE INFOCOM 2010*, San Diego, March 2010, pp. 1828-1836.
[11] Z. Q. Xia and Z. G. Chen. "Wireless Multi-path Inter-session Network Coding Game," *Journal of*

Information and Computational Science, vol. 7, no. 13, December 2010, pp. 2763-2770.

[12] K. Kim and N. Venkatasubramanian. "Assessing the impact of geographically correlated failures on overlay-based data dissemination," *IEEE GLOBECOM 2010*, December 2010, pp. 1–5.

[13] F. Bernhard, E. Alireza, V. D. V. Dimitri, and M. Torsten, "Efficient volume rendering on the body centered cubic lattice using box splines," *Computers & Graphics*, 2010, vol. 3, pp. 409-423.

[14] B. Csebfalvi, "An Evaluation of Prefiltered B-Spline Reconstruction for Quasi-Interpolation on the Body-Centered Cubic Lattice," *IEEE Transactions on Visualization and Computer Graphics*, 2010, vol. 16, pp. 499 – 512.