

Software Quality Metrics for Object-Oriented Environments

AUTHORS:

Dr. Linda H. Rosenberg
Unisys Government Systems
Goddard Space Flight Center
Bld 6 Code 300.1
Greenbelt, MD 20771 USA

Lawrence E. Hyatt
Software Assurance Technology Center
Goddard Space Flight Center
Bld 6 Code 302
Greenbelt, MD 20771 USA

I. INTRODUCTION

Object-oriented design and development are popular concepts in today's software development environment. They are often heralded as the silver bullet for solving software problems. While in reality there is no silver bullet, object-oriented development has proved its value for systems that must be maintained and modified. Object-oriented software development requires a different approach from more traditional functional decomposition and data flow development methods. This includes the software metrics used to evaluate object-oriented software.

The concepts of software metrics are well established, and many metrics relating to product quality have been developed and used. With object-oriented analysis and design methodologies gaining popularity, it is time to start investigating object-oriented metrics with respect to software quality. We are interested in the answer to the following questions:

- What concepts and structures in object-oriented design affect the quality of the software?
- Can traditional metrics measure the critical object-oriented structures?
- If so, are the threshold values for the metrics the same for object-oriented designs as for functional/data designs?
- Which of the many new metrics found in the literature are useful to measure the critical concepts of object-oriented structures?

II. METRIC EVALUATION CRITERIA

While metrics for the traditional functional decomposition and data analysis design approach measure the design structure and/or data structure independently, object-oriented metrics must be able to focus on the combination of function and data as an integrated object [1]. The evaluation of the utility of a metric as a quantitative measure of software quality was based on the measurement of a software quality attribute. The metrics selected, however, are useful in a wide range of models. The object-oriented metric criteria, therefore, are to be used to evaluate the following attributes:

- Efficiency - Are the constructs efficiently designed?
- Complexity - Could the constructs be used more effectively to decrease the architectural complexity?
- Understandability - Does the design increase the psychological complexity?

- Reusability - Does the design quality support possible reuse?
- Testability/Maintainability - Does the structure support ease of testing and changes?

Whether a metric is “traditional” or “new”, it must be effective in measuring one or more of these attributes. As each metric is presented, we will briefly discuss its applicability.

The SATC’s approach to identifying 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. The suggested 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 software.

We support the use of three traditional metrics and present six additional metrics specifically for object-oriented systems. The SATC has found that there is considerable disagreement in the field about software quality metrics for object-oriented systems [1,4]. 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 software quality attributes have been validated [1,4,8,9] Preceding each metric, a brief description of the object-oriented structure is given. Each metric is then described, interpretation guidelines given, and the applicable quality attributes listed.

III. TRADITIONAL METRICS

A. Methods

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. Methods reflect how a problem is broken into segments and the capabilities other classes expect of a given class. Two traditional metrics are discussed here: cyclomatic complexity and size (line counts).

- **METRIC 1: Cyclomatic Complexity (CC)**
Cyclomatic complexity (McCabe) is used to evaluate the complexity of an algorithm in a method. A method with a low cyclomatic complexity is generally better, although it may mean that decisions are deferred through message passing, not that the method is not complex. Cyclomatic complexity cannot be used to measure the complexity of a class because of inheritance, but the cyclomatic complexity of individual methods can be combined with other measures to evaluate the complexity of the class. In general, the cyclomatic 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 [2,3,4,5,8].

- **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. [2,4,5,8]

- **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% is most effective. Since comments assist developers and maintainers, this metric is used to evaluate the attributes of Understandability, Reusability, and Maintainability [6].

IV. OBJECT-ORIENTED SPECIFIC METRICS

As discussed, 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 software analysis package being used to collect the metrics.

A Class

A class is a template from which objects can be created. This set of objects share a common structure and a common behavior 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.

A.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 cyclomatic 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 [1,4,5,7].

A.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 cardinality 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 [1,4,5,7].

A.3 Cohesion

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 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 [1,2,4,5,7].

A.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. The next object-oriented metric measures coupling strength.

- **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 [1,2,3,4,5,7].

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.

- **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 [1,2,4,5,7].

- 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 [1,4,5,7].

V. SUMMARY

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 be developed, based on the effect of the metric on desirable software qualities.

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. Hudli, R., Hoskins, C., Hudli, A., "Software Metrics for Object-oriented Designs", IEEE, 1994.
3. Lee, Y., Liang, B., Wang, F., "Some Complexity Metrics for Object-Oriented Programs Based on Information Flow", *Proceedings: CompEuro*, March, 1993, pp. 302-310.
4. Lorenz, Mark and Kidd, Jeff, *Object-Oriented Software Metrics*, Prentice Hall Publishing, 1994.
5. McCabe & Associates, *McCabe Object-Oriented Tool User's Instructions*, 1994.
6. Rosenberg, Linda and Hyatt, Lawrence, Set Laboratories, *UX Metrics*, 1994.

7. Sharble, Robert, and Cohen, Samuel, "The Object-Oriented Brewery: A Comparison of Two Object-Oriented Development Methods", *Software Engineering Notes*, Vol 18, No 2., April 1993, pp 60 -73.
 8. Tegarden, D., Sheetz, S., Monarchi, D., "Effectiveness of Traditional Software Metrics for Object-Oriented Systems", *Proceedings: 25th Hawaii International Conference on System Sciences*, January, 1992, pp. 359-368.
- Williams, John D., "Metrics for Object-Oriented Projects", *Proceedings: ObjectExpoEuro Conference*, July, 1993, pp. 13-18.