
# Principal Components of Orthogonal Object-Oriented Metrics (323-08-14)

White Paper Analyzing Results of NASA Object-Oriented Data

**Submitted By: Victor Laing**
**And**
**Charles Coleman, Manager, SATC**

October 12, 2001

**Technical POC:** Dr. Linda Rosenberg

**Phone #:** 301-286-0087
**Fax #:** 301-286-1701
**Email:** Linda.Rosenberg@gsfc.nasa.gov
**Mail Code:** 304

**Administrative POC:** Dennis Brennan

**Phone #:** 301-286-6582
**Fax #:** 301-286-1667
**Email:** Dennis.Brennan@gsfc.nasa.gov
**Mail Code:** 300

# ABSTRACT

This report presents the results of Task 323-08-14, Principal Components of Orthogonal Object-Oriented Metrics, performed by the Software Assurance Technology Center (SATC) at NASA Goddard Space Flight Center. The task developed an approach to formulating a set of Orthogonal Object-Oriented metrics. The research is intended to find a way to produce cheaper and higher quality software. The Orthogonal Object-Oriented set of metrics will be selected from the core set of metrics that the SATC uses for code analysis.

# EXECUTIVE SUMMARY

The development of a large software system is a time and resource-consuming activity. Even with the increasing automation of software development activities, resources are still scarce. There is also a great interest in software metrics due to their potential for use as a cost saving device.

This paper presents the research results of the Software Assurance Technology Center (SATC) at NASA Goddard Space Flight Center, to develop an approach to formulating a set of Orthogonal Object-Oriented metrics. The research is intended to find a way to produce cheaper and higher quality software. The Orthogonal Object-Oriented set of metrics will be selected from the core set of metrics that the SATC uses for code analysis.

The set of Orthogonal Object-Oriented metrics obtained in the study was applied to three real world industrial strength object-oriented systems to predict their overall quality. The level of quality found in these three systems is classified into three types: low, medium, and high. The classification obtained for the systems using the Orthogonal Object-Oriented metrics set were validated with the SATC approach used for their code analysis for identifying code quality.

**Table of Contents**

# 1. Introduction

The Software Assurance Technology Center (SATC) at NASA Goddard Space Flight Center is currently conducting research on an approach to formulating a set of Orthogonal Object-Oriented metrics. The research is intended to find a way to produce cheaper and higher quality software. It was determined by the SATC that there is a lack of research being conducted in the area of Orthogonal Object-Oriented metrics. The potential benefit of this research can be applied to both NASA and industry.

Object-Oriented Programming (OOP) is a programming paradigm that is based on abstractions of object types in the application [HUD]. The key difference between object-oriented programming and structured programming is that the former identifies the object types in the applications, while the latter models the applications as a set of functions.

There is a general shift in the industry from the structured (traditional) programming and development environment to an object-oriented paradigm, and NASA is no exception in adhering to this shift. If organizations wish to make a successful change, they need the appropriate metrics for this new paradigm. One such metrics suite was proposed by Chidamber and Kemerer [CHI].

There is also a great deal of interest in software metrics due to their potential for use in procedures to control cost of system development and maintenance activities [TEG]. However, metrics programs are sometimes viewed as being costly with little return on investment. One way of reducing the cost of a metrics program, increasing efficiency, and decreasing the intrusiveness caused by the metrics program is to measure "less". This is feasible only if we can obtain at least the same level of accuracy with the reduced information obtained by measuring "less". A good candidate for accomplishing this task is Orthogonal Object-Oriented metrics, if one is working in an object-oriented environment.

The set of Orthogonal Object-Oriented metrics proposed in this paper is designed to evaluate key features of object-oriented design such as encapsulation, inheritance, and polymorphism. This evaluation is used to classify the quality level of object-oriented software systems.

The remainder of the paper is organized as follows: Section 2 first provides an overview of both traditional and object-oriented metrics. This section also defines and discusses the concept of Orthogonal Object-Oriented metrics. The theoretical framework for the Orthogonal Object-Oriented metrics set is developed in Section 3. An empirical investigation is conducted on three industrial strength object-oriented metrics to validate the metrics set in Section 4. Our conclusions from this empirical study and the research results are given in Section 5. Future research directions are also presented in this section.

# 2. Background

Traditional and object-oriented programming are fundamentally different and therefore different metrics are needed for their evaluation. According to Moreau [MOR87], [MOR89], [MOR90], traditional metrics are inappropriate for object-oriented systems. However, research conducted by Tegarden [TEG] has shown that a combination of both traditional and object-oriented metrics may give the best results when analyzing object-oriented systems with respect to their overall quality.

In Subsection 2.1 we define three traditional metrics, that are popular with practitioners and researchers. The Chidamber and Kemerer (CK) metrics suite for object-oriented design are given in Subsection 2.2. Subsection 2.3 defines and explains the concept of Orthogonal Object-Oriented metrics. The SATC uses most of the metrics listed below or a modification of them (see [ROS95], [ROS97], [ROS98]).

## 2.1 Traditional Metrics

Traditional metrics have been applied to the measurement of software complexity of structured systems since 1976 [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. Usually 20% indicates adequate commenting for C or Fortran code [ROS95]. A high CP value facilitates in maintaining a system.

## 2.2 Object-Oriented Metrics

In 1994 Chidamber and Kemerer [CHI] proposed a now widely accepted suite of metrics for an object-oriented system. Basili validated the metrics suite in 1996 [BAS] and Tang in 1999 [TAN]. The six object-oriented metrics are listed below.

*Weighted Methods Per Class (WMC)*: WMC measures the complexity of an individual class. Two different approaches are used to calculate the WMC metric. The first uses the sum of the complexity of each method contained in the class. The second approach assigns a complexity of 1 for each method in the class and then sums the result. This is equivalent to using the number of methods per class as a measure for WMC [CHI]. The number of methods and complexity of methods involved is a direct predictor of how much time and effort is required to develop and maintain the class.

*Depth of Inheritance Tree of a Class (DIT)*: DIT is defined as the length of the longest path of inheritance ending at the current module [CHI]. In cases involving multiple inheritances, the DIT will be the maximum length from the node to the root of the tree [CHI]. The deeper the inheritance tree for a class, the harder it might be to predict its behavior due to the interaction between the inherited features and new features. However, the deeper a particular class is in the hierarchy, the greater the potential for reuse of inherited methods.

*Number of Children (NOC)*: NOC represents the number of immediate subclasses subordinated to a class in the class hierarchy [CHI]. A moderate value for NOC indicates scope for reuse and high values may indicate an inappropriate abstraction in the design. Classes with a large number of children have to provide more generic service to all the children in various contexts and must be more flexible, a constraint that can introduce more complexity into the parent class.

*Coupling Between Objects (CBO)*: CBO is defined as the count of the number of other classes to which it is coupled [CHI]. A class is coupled to another class if it uses the member method and/or instance variables of the other class. Excessive coupling indicates weakness of class encapsulation and may inhibit reuse. High coupling also indicates that more faults may be introduced due to inter-class activities.

*Response for a Class (RFC)*: RFC gives the number of methods that can potentially be executed in response to a message received by an object of that class [CHI]. If a large number of methods can be invoked in response to a message, the testing and debugging of the class becomes more complicated since it requires a greater level of understanding required on the part of the tester.

*Lack of Cohesion in Methods (LOCM)*: LOCM counts the number of method pairs whose similarity is 0 minus the count of method pairs whose similarity is not zero. The larger the number of similar methods in a class the more cohesive the class is [CHI].

Cohesiveness of methods within a class is desirable, since it promotes encapsulation and lack of cohesion implies classes should probably be split into two or more subclasses.

### 2.3 Orthogonal Object-Oriented Metrics

This subsection describes what it means for two or more object-oriented metrics to be orthogonal. The main focus of this study is to produce a minimal set of Orthogonal Object-Oriented metrics capable of analyzing code quality with the same degree of accuracy as afforded by a metrics set of a significantly larger cardinality.

*Orthogonal*: Orthogonality is a measure of intrinsically different characteristics of the code, therefore any correlation 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.

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

## 3. Theoretical Framework

This section provides the foundation and justification of a theoretical framework for developing an Orthogonal Object-Oriented metrics suite. Subsection 3.1 gives an overview of the more important aspects of the object-oriented paradigm. The CK suite of object-oriented metrics is reduced to a single equation and one standalone metric in Subsection 3.2. A discussion is presented in Subsection 3.3 on how to apply this reduced CK metrics suite.

### 3.1 Overview of the Object-Oriented Paradigm

Object-oriented modeling 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 behavior in a single entity [RUM]. 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 [TEG]. These three aspects of the object-oriented paradigm are described below.

*Encapsulation*: Encapsulation consists of separating 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 use 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, there can be different operations to pay employees based on the employee's object type, e.g., part-time, hourly, or salaried. Each type of employee object can have its own customized compute-pay operation. When a new type of employee is created, e.g., student, the programmer simply creates a new type of employee object and a new implementation of compute-pay for the new type of employee. When an instance of student receives the message to compute-pay, it uses the operation defined in the new object to perform the calculation. The compute-pay operations of the other types of employees are not affected by the payment operations required for the student. In contrast, structured systems often have all pay operations contained in one program. The program must be capable of differentiating between the different types of employees and applying the appropriate operation. A modification to a new type of employee typically requires existing structured code to be changed.

*Inheritance*: Inheritance is a reuse mechanism that allows programmers to define objects incrementally by reusing previously defined objects as the basis for new objects. For example, when defining a new type of employee (e.g., student), the new employee type can inherit the characteristics common to all employees (e.g., name, address), from a generic type of employee. In this approach, the programmer needs only to be concerned with the difference between student employees and generic employees. Structured systems do not have an inheritance mechanism as part of their formal specification.

### 3.2 A Reduced CK Metrics Suite

In Section 2 we listed the complete CK suite of object-oriented metrics, however they were not rigorously defined. A subset of the suite is formally defined (name, definition, and theoretical basis) in this subsection in order to develop the reduced suite. See [CHI] for the formal definitions of the complete set of the CK suite of object-oriented metrics.

*Metric 1*: *Weighted Methods Per Class (WMC)*

*Definition 1*: Consider a class $C_1$, with methods $M_1$, …, $M_n$ that are defined in the class. Let $c_1$, …, $c_n$ be the complexity of the methods. Then:

$$\text{WMC} = \sum_{i=1}^{n} c_i.$$

If all method complexities are considered to be unity, then WMC = $n$, the number of methods.

*Theoretical Basis 1*: WMC relates directly to Bunge's [BUN77], [BUN79] definition of complexity of a thing, because methods are properties of object classes and complexity is determined by the cardinality of its set of properties. The number of methods is therefore a measure of the class definition as well as attributes of a class, because attributes correspond to properties.

*Metric 2*: *Coupling Between Objects (CBO)*

*Definition 2*: CBO for a class is the count of the number of other classes to which it is coupled.

*Theoretical Basis 2*: CBO relates to the notion that an object is coupled to another object if one of them acts on the other, i.e., methods of one use methods or instance variables of another. Since objects of the same class have the same properties, two classes are coupled when methods declared in one class use methods or instance variables defined by the other class.

*Metrics 3*: *Response for a Class (RFC)*

*Definition 3*: *RFC* = |*RS*| where *RS* is the response set for the class.

*Theoretical Basis 3*: The response set for the class can be expressed as

$$RS = \{M\} U_{all\ i} \{R_i\}$$

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

The response set of a class is a set of methods that can potentially be executed in response to a message received by an object of that class. The cardinality of this set is a measure of the attributes of objects in the class. Since it specifically includes methods called from outside the class, it is also a measure of the potential communication between the class and other classes.

The formal definitions above are now used to construct an equation relating two out of the three more important aspects (encapsulation, polymorphism, and inheritance) of the object-oriented paradigm namely, encapsulation and polymorphism. From Definition 3 we have:

$$RFC = NLM + NRM \qquad (1)$$

where NLM = number of local methods in a class and NRM = number of remote methods called from a class. Definition 1 stated that if all the method complexities in a class are considered to be unity, then WMC = *n*, the number of methods in the class, which gives NLM = WMC [CHI].

It was mentioned in Section 2 that excessive coupling between objects indicates weakness of class encapsulation and may inhibit reuse. However, some coupling between objects is necessary for objects to be able to interact with each other. Ideally, objects should be coupled as loosely as possible in order to promote encapsulation and reusability. This is accomplished by having objects interact with which other exclusively through their interface. There are other types of coupling. The CBO metric defined in Definition 2 describes objects that are tightly coupled; the objects are accessed internally though remote methods calls (NRM) to other object methods or instance variables.

Because tight coupling between objects is undesirable, we shall use this fact as one of the cornerstones in identifying low quality software. On average, the number of remote method calls is much larger than the number of instance variables accessed from one object to the next. Thus, under tight coupling of objects the number of remote methods calls approximates the measure of coupling between objects that is NRM ≈ CBO.

Substituting the metrics WMC and CBO for NLM and NRM respectively into Equation 1 (RFC = NLM +NRM) gives:

$$RFC = WMC + CBO \qquad (2).$$

Equation 2 and the DIT metrics shall be used to identify low quality software in Section 4, Empirical Investigation.

### 3.3 Applying the Reduced Metrics Suite

There are three fundamental aspects of the object-oriented paradigm, namely encapsulation, polymorphism, and inheritance. The equation RFC = WMC + CBO captures both encapsulation and polymorphism and their relationship to each other. High values for WMC and CBO indicate low encapsulation [HUD], [LOR93], [LOR94] a class may be implementing too much of a system's functionality or may be coupled too tightly. The RFC metric measures polymorphism by recording the number of remote method calls by the class, for example, virtual methods in the C++ programming language is a direct implementation of the concept of polymorphism. It is clear that the DIT metric quantifies the inheritance aspect of the object-oriented paradigm and should need no further explanation.

Examining equation RFC = WMC + CBO more closely suggests that if WMC and CBO increases then RFC also increases and if WMC and CBO decreases, RFC also decreases. If objects in a system are loosely coupled (one indicator of high quality code) then the CBO metric will be low and increases in RFC are due to WMC. That is, including more methods in a class increases the RFC metric but since coupling is loose, the CBO metric should not significantly increase. On the other hand, if objects in a system are tightly coupled (one indicator of low quality code) then the CBO metric will be high and increases in RFC would be due to both CBO and WMC. Different researchers like Kidd, Lorenz, [LOR94] and Rosenberg [ROS97] give similar preferred highest values for these object-oriented metrics along with values for the DIT metric also.

Thus, as coupling between objects increases, the equation RFC = NLM + NRM approaches the equation RFC = WMC + CBO. This can be used to identify low quality object-oriented systems and with the DIT metrics capture the three more important aspects of the object-oriented paradigm: encapsulation, polymorphism, and inheritance.

This section provides theoretical justification for suggesting the completeness and sufficiency of a minimal set of object-oriented metrics for analyzing the quality of an object-oriented system, where some of the metrics were related in the form of an equation. The next section applies this reduced set of object-oriented metrics to three real world projects to measure their overall quality and compare the results with the results obtained from using a larger set of both traditional and object-oriented metrics.

# 4. Empirical Investigation

This section applies the results obtained in the study to three industrial strength software systems. The outcome is validated with previous results obtained from extensive full-scale code analysis performed by the SATC on the same three systems.

Subsection 4.1 describes the software applications used in validating the reduced object-oriented metrics set in detail. The statistical terminologies used in the investigation are defined and discussed in Subsection 4.2 and the statistical analysis is conducted in Subsection 4.3.

## 4.1 Descriptions of the Applications

The three applications used in this empirical study to validate the reduced object-oriented metrics set are industrial strength software. Two of the applications were NASA systems and one was a commercial product. We labeled the applications as: System A, System B, and System C. System A was the commercial software implemented in Java and consisted of approximately 50,000 lines of code and had 46 classes. One of the NASA software applications was also implemented in Java and consisted of approximately 300,000 lines of code and contained 1,000 classes. We labeled this application System B. The last application, System C, was also a NASA product approximately consisting of 500,000 lines of code distributed over 1,617 classes. System C was implemented in the C++ programming language.

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: Summary of Applications used to Validate Reduced CK Metrics Set**

| System | A | B | C |
|---|---|---|---|
| **Lines of Code** | 50k | 300k | 500k |
| **Number of Classes** | 46 | 1000 | 1617 |
| **Language** | Java | Java | C++ |
| **Type of Application** | Commercial Software | NASA Software | NASA Software |
| **Code Construct** | Object-Oriented | Excellent Object-Oriented | Good Object-Oriented |
| **Quality** | Low | High | Medium |

## 4.2 Definitions for Statistical Analysis

In order to investigate the correlations and relationships between the object-oriented metrics and software quality we conducted a correlation and a multiple linear regression analysis. Definitions and discussions on the terminologies used in the correlation and regression analysis are provided in this section.

A multiple linear regression model can be defined as

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p + \varepsilon \qquad (3).$$

The various components of the regression model above along with other statistical terminologies used in the paper are listed below:

*Independent Variable ($X_i$'s)*: The independent variable in an experiment is the variable that is systematically manipulated by the investigator. In most experiments, the investigator is interested in determining the effect that one variable; has on one or more of the other variables. In the regression model [3] the $X_i$'s denote the independent variables.

*Dependent Variable (Y)*: The dependent variable in an experiment is the variable that the investigator measures to determine the effect of the independent variable. In the regression equation [3] variable $Y$ denotes the dependent variable.

*Random Error ($\varepsilon$)*: The variation in the observed data that is not accounted for by the model. In the regression model [3] the random error is denoted by $\varepsilon$.

*Coefficients ($\beta_i$'s)*: The estimated multiple linear regression coefficients measure the respective independent variable's contribution on the dependent variable. The larger the absolute coefficient values, the larger (positive or negative according to the sign) the impact of the independent variable on the dependent variable. In the regression model [3] $\beta_0$ represents the constant term.

*Linear Correlation Coefficient (r)*: The linear correlation coefficient expresses quantitatively the magnitude and direction of the linear relationship between two variables. A correlation coefficient can vary from +1 to −1. The sign of the coefficient

tells us whether the relationship is positive or negative. The numerical part of the correlation coefficient describes the magnitude of the correlation. The higher the number, the greater the correlation.

*The Statistical Significance (p-value)*: Represents the degree of accuracy of coefficient estimation. More specifically, the p-value represents the probability of error that is involved in accepting observed results as valid. The larger the p-value, the less believable the estimated impact of the explanatory independent variable (OO metric). In our study we used 0.05 as the significance threshold.

*The Goodness of Fit ($R^2$)*: The goodness of fit is an indicator of how well the model fits the data. The higher the value of $R^2$, the more accurate the model is.

### 4.3 Statistical Analysis

Figures 1 – 3 show the distributions of the reduced CK object-oriented metrics suite namely, the RFC, CBO, WMC, and DIT metrics for the three systems. The percentage of classes is on the x-axis and the metrics values on the y-axis. The distributions for Systems A, B, and C included 46 Java, 1,000 Java, and 1,617 C++ classes respectively.

The shapes of the distributions for all the metrics are similar between systems except for the DIT metric. The shape of the DIT metric for Systems B and C are similar. However, System A exhibits a different distribution from both Systems B and C. The distribution for System A shows that over 60% of the classes in that system had a DIT metric of 0, suggesting a lack of reuse via inheritance.

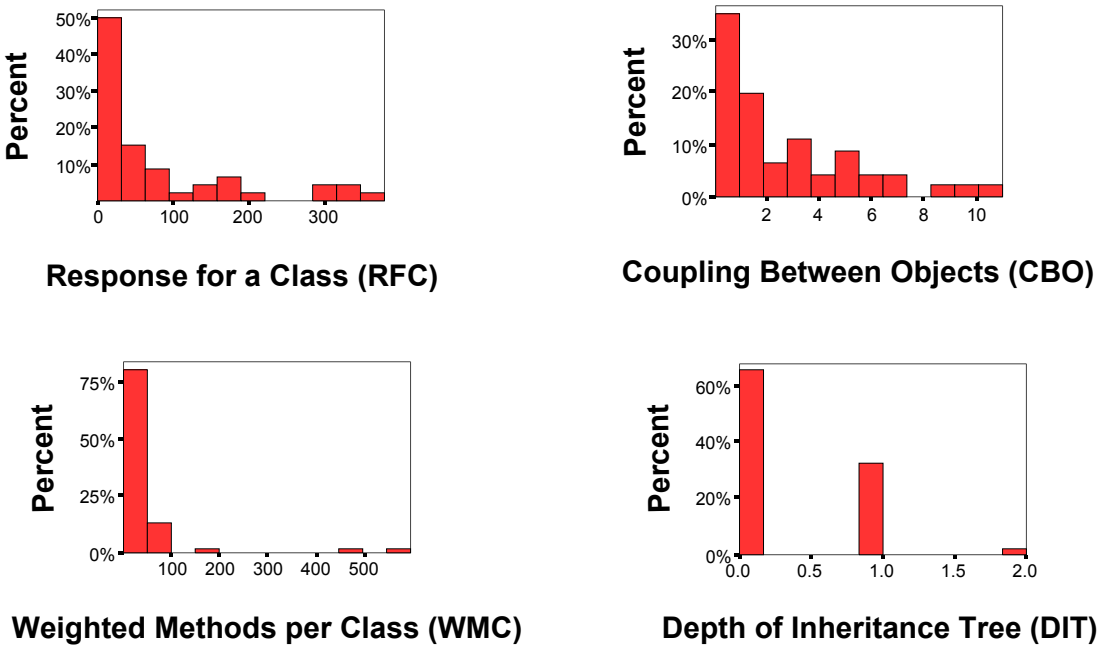


**Figure 1: Distributions of the Reduced CK Metrics Set for System A**

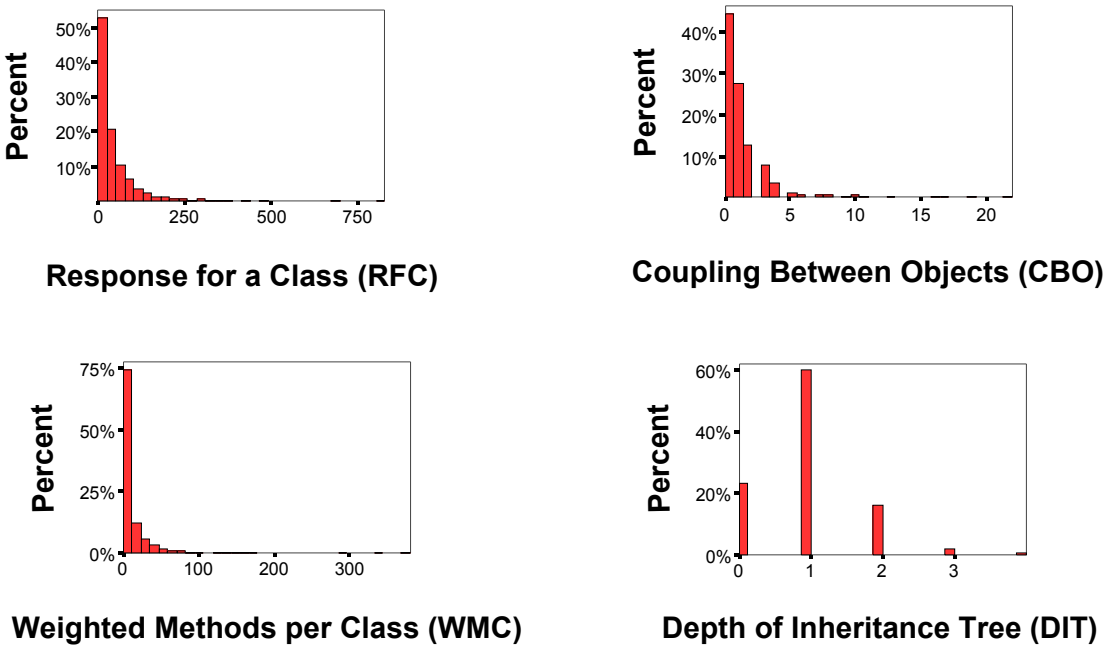


**Figure 2: Distributions of the Reduced CK Metrics Set for System B**

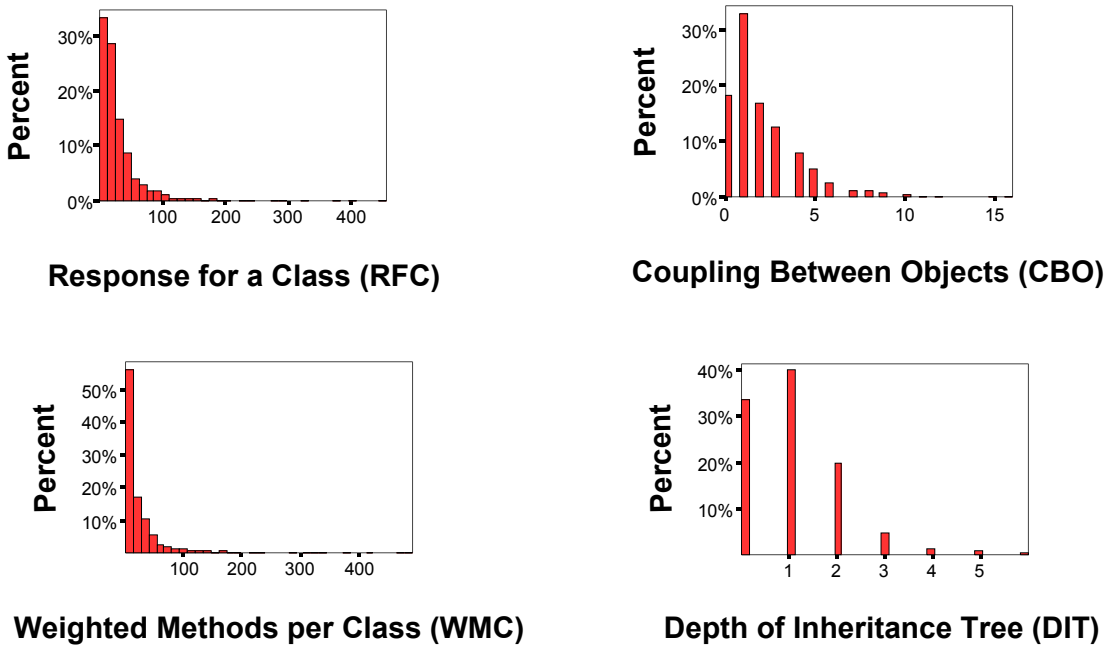


**Figure 3: Distributions of the Reduced CK Metrics Set for System C**

The descriptive statistics and the correlations between the metrics for each system are given in Tables 2 – 7. The values in bold are the mean values of the reduced metrics set. The descriptive statistics for Systems A, B, and C are included on 46 Java, 1,000 Java, and 1,617 C++ classes respectively. The descriptive statistics tables show that the mean values for the CBO, RFC, and WMC metrics for System B and C are lower than the values for System A indicating higher code quality for System B and C over System A.

In the descriptive statistics tables the values are provided for the full set of the CK metrics suite for completeness although we only used the reduced set highlighted in bold. The correlation results (tables) are discussed at the end of this section in conjunction with the regression models.

**Table 2: Descriptive Statistics for System A**

| | Minimum | Maximum | Mean | Std. Deviation |
|---|---|---|---|---|
| **CBO** | 0 | 11 | **2.48** | 2.93 |
| **LCOM** | 0 | 3804 | 447.65 | 1015.03 |
| **RFC** | 0 | 381 | **80.39** | 101.91 |
| **NOC** | 0 | 2 | 0.07 | 0.32 |
| **DIT** | 0 | 2 | **0.37** | 0.53 |
| **WMC** | 0 | 596 | **45.70** | 110.95 |

**Table 3: Correlation Analysis for System A**

| | CBO | LCOM | RFC | NOC | DIT | WMC |
|---|---|---|---|---|---|---|
| **CBO** | 1 | 0.36 | **0.83** | -0.17 | -0.20 | **0.43** |
| **LCOM** | | 1 | 0.66 | -0.09 | -0.09 | 0.03 |
| **RFC** | | | 1 | -0.16 | -0.22 | **0.47** |
| **NOC** | | | | 1 | -0.01 | -0.08 |
| **DIT** | | | | | 1 | -0.03 |
| **WMC** | | | | | | 1 |

**Table 4: Descriptive Statistics for System B**

| | Minimum | Maximum | Mean | Std. Deviation |
|---|---|---|---|---|
| **CBO** | 0 | 22 | **1.25** | 2.01 |
| **LCOM** | 0 | 5444 | 78.34 | 284.72 |
| **RFC** | 0 | 827 | **43.84** | 65.31 |
| **NOC** | 0 | 21 | 0.35 | 1.66 |
| **DIT** | 0 | 4 | **0.97** | 0.69 |
| **WMC** | 0 | 381 | **11.10** | 26.12 |

**Table 5: Correlation Analysis for System B**

| | CBO | LCOM | RFC | NOC | DIT | WMC |
|---|---|---|---|---|---|---|
| **CBO** | 1 | 0.10 | **0.28** | -0.03 | 0.34 | **0.11** |
| **LCOM** | | 1 | 0.76 | -0.01 | 0.10 | 0.68 |
| **RFC** | | | 1 | -0.09 | 0.16 | **0.75** |
| **NOC** | | | | 1 | -0.10 | -0.07 |
| **DIT** | | | | | 1 | 0.05 |
| **WMC** | | | | | | 1 |

**Table 6: Descriptive Statistics for System C**

| | Minimum | Maximum | Mean | Std. Deviation |
|---|---|---|---|---|
| **CBO** | 0 | 16 | **2.09** | 2.05 |
| **LCOM** | 0 | 3281 | 113.94 | 266.03 |
| **RFC** | 0 | 457 | **28.60** | 36.24 |
| **NOC** | 0 | 116 | 0.39 | 3.22 |
| **DIT** | 0 | 6 | **1.02** | 0.98 |
| **WMC** | 0 | 492 | **23.97** | 40.26 |

**Table 7: Correlation Analysis for System C**

| | CBO | LCOM | RFC | NOC | DIT | WMC |
|---|---|---|---|---|---|---|
| **CBO** | 1 | 0.25 | **0.35** | 0.00 | 0.33 | **0.26** |
| **LCOM** | | 1 | 0.61 | 0.05 | -0.02 | 0.48 |
| **RFC** | | | 1 | 0.01 | -0.00 | **0.83** |
| **NOC** | | | | 1 | -0.01 | 0.00 |
| **DIT** | | | | | 1 | -0.10 |
| **WMC** | | | | | | 1 |

Tables 3, 5, and 7 present the correlation analysis for the CK object-oriented metrics suite over the various software systems. The values in bold are the mean values of the reduced metrics set excluding the DIT metric. A correlation analysis was conducted for Systems A, B, and C where the systems had 46 Java, 1,000 Java, and 1,617 C++ classes respectively. Table 8 summarizes the results of the correlation analysis for the reduced metrics set over the various software systems. The columns list the correlation values for each pair of metrics in the reduced metrics set and the rows list the system. In the table Metric 1 x Metric 2 = correlation between Metric 1 and Metric 2.

Examining Table 8 shows that for System A all of the metrics are highly correlated with each other, with CBO and RFC being the most significantly correlated. This suggests low quality code because RFC increases due to tight coupling in the system and not because of the introduction of methods into the system. Equation 2 shows that RFC = WMC + CBO. The opposite is true for System B and C, that is, the correlation between CBO and RFC is low and between CBO and WMC. However, the correlation between RFC and WMC is high. This shows that RFC increases as the number of methods increases in these applications, using Equation 2. These results show that Systems B and C consist of loosely coupled objects, implying high quality code with System B having high quality. Similar findings were obtained from the regression analysis.

**Table 8: Correlation Analysis Summary**

| | CBOxRFC | CBOxWMC | RFCxWMC |
|---|---|---|---|
| **System A** | 0.83 | 0.43 | 0.47 |
| **System B** | 0.28 | 0.11 | 0.75 |
| **System C** | 0.35 | 0.26 | 0.83 |

The multiple linear regression model listed in Equation 4 was fitted to the reduced set of metrics for Systems A, B, and C respectively and the results are given in Table 9.

$$RFC = \beta_{WMC} WMC + \beta_{CBO} CBO + Constant \qquad (4)$$

The standardized coefficients ($\beta_i$'s) for the regression model of each system were all significant at the 0.01 level and each having a p-value of 0.000, except for the coefficient of the WMC metric for System A where $\beta_{WMC}$ and *Constant* had a p-value of 0.160 and

0.466 respectively at a level of significance of 0.01. These results reinforce the findings of the correlation analysis suggesting that objects in System A are tightly coupled and increases in the RFC metric are due to increases in CBO and not WMC, implying low quality code.

On the other hand the results for Systems B and C are the reverse. This can be explained by observing the magnitudes of the standardized coefficients for both of the independent variables WMC and CBO in the model since they were both significant. The magnitude of the coefficient for the independent variable WMC is approximately four times larger than that of CBO for Systems B and C. This implies that increases to the RFC metric are due to an increase in the number methods in a class and not to an increase in coupling. Thus Systems B and C consist of loosely coupled objects indicating high quality code.

**Table 9: Regression Analysis Summary**

| Estimated Parameters | System A | System B | System C |
|---|---|---|---|
| $\beta_{WMC}$ (Unstandardized) | 0.120 | 1.831 | 0.712 |
| $\beta_{CBO}$ (Unstandardized) | 26.985 | 6.471 | 2.623 |
| *Constant* (Unstandardized) | 8.036 | 15.427 | 6.039 |
| $\beta_{WMC}$ (Standardized) | 0.131 | 0.732 | 0.792 |
| $\beta_{CBO}$ (Standardized) | 0.777 | 0.200 | 0.148 |
| $R^2$ | 0.708 | 0.608 | 0.708 |

The quality of the code for System B is of a higher quality than System C with respect to the magnitudes of the standardized coefficients in the regression models. Also the fit of the regression model over each system was appropriate since each had an $R^2$ of approximately 70%.

In the correlation and regression analysis the three main aspects of the object-oriented paradigm, encapsulation, polymorphism, and inheritance were not explicitly mentioned in relation to code quality. However, the coupling of objects in the systems was discussed at length in conjunction with code quality using the CBO metric and this metric measures encapsulation. Polymorphism of a system is measured indirectly by the metrics RFC and WMC, and these metrics were related in Equations 2 and 4. Although the case of inheritance was not dealt with in the correlation and regression analysis, the inheritance over the systems was evaluated using the DIT metric, as shown in the descriptive statistics table for each system.

## 5. Conclusions and Future Work

We have constructed a simple and easy 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.

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 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 low quality software, System B was high, and System C was medium.

The reduced CK metrics set approach is very promising. A word of caution is prudent at this point: the reduced metrics set approach is not a silver bullet. The approach is more applicable to identifying 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.

## 6. References


[BAS] Basili, V. R., Briand L. C., and Melo, W. L. "A Validation of Object-Oriented Design Metrics as Indicators", *IEEE Transactions on Software Engineering*, 22(10):551-761, October 1996.

[BUN77] Bunge, M., *Treatise on Basic on Philosophy: Ontology I: The Furniture of the World*, Boston: Riedel, 1977.

[BUN79] Bunge, M., *Treatise on Basic on Philosophy: Ontology II: The World of the Systems*, Boston: Riedel, 1979.

[CHI] Chidamber, S. R. & Kemerer, C. F., "A Metrics Suite for Object Oriented Design", *IEEE Transactions on Software Engineering*, Vol. 20, #6, June 1994.

[HAR98] Harrison, R., Counsell, S. J., and Nithi, R. V., "An Evaluation of the MOOD Set of Object-Oriented Software Metrics", *IEEE Transaction on Software Engineering*, Vol. 24, No. 6, June 1998.

[HIT] Hitz, M. and Montazeri, B. "Chidamber and Kemerer's Metrics Suite: A Measurement Theory Perspective", *IEE Transaction on Software Engineering*, Vol. 22, No. 4, April 1996.

[HUD] Hudli, R., Hoskins, C., Hudli, A., "Software Metrics for Object Oriented Designs", *IEEE Transactions on Software Engineering*, 1994.

[LAC] Lacovara , R.C., and Stark G. E., "A Short Guide to Complexity Analysis, Interpretation and Application", May 17, 1994. http://members.aol.com/GEShome/complexity/Comp.html

[LIW] Li, W, Henry, S., et. al., "Measuring Object Oriented Design," *Journal of Object Oriented Programming*, Vol. 8, #4, July 1995.

[LOR93] Lorenz, Mark, *Object Oriented Software Development*, Prentice Hall, 1993.

[LOR94] Lorenz, Mark & Kidd Jeff, *Object-Oriented Software Metrics*, Prentice Hall, 1994.

[MCC99] McCabe and Associates, *Using McCabe QA 7.0*, 1999, 9861 Broken Land Parkway 4th Floor Columbia, MD 21046.

[MCC76] McCabe, T. J., "A Complexity Measure", *IEEE Transactions on Software Engineering*, SE-2(4), pages 308-320, December 1976.

[MOR87] Moreau, D. R., "A Programming Environment Evaluation Methodology for Object-Oriented Systems", Ph.D. Dissertation, University of Southwestern Louisiana, 1987.

[MOR89] Moreau, D. R., and Dominick, W. D., "Object-Oriented Graphical Information Systems: Research Plan and Evaluation", *Journal of Systems and Software*, vol. 10, pp. 23-28, 1989.

[MOR90] Moreau, D. R., and Dominick, W. D., "A Programming Environment Evaluation Methodology for Object-Oriented Systems: Part I – The Methodology", *Journal of Object-Oriented Programming*, vol. 3, pp. 38-52, 1990.

[ROS97] Rosenberg, L., "Metrics for Object-Oriented Environment", EFAITP/AIE Third Annual Software Metrics Conference, 1997.

[ROS98] Rosenberg, L., and Hammer, T., "Metrics for Quality Assurance and Risk Assessment", *Proc. Eleventh International Software Quality Week*, San Francisco, CA, 1998.

[ROS95] Rosenberg, L., and Hyatt, L., "Software Quality Metrics for Object-Oriented System Environments", Software Assurance Technology Center, Technical Report SATC-TR-95-1001, NASA Goddard Space Flight Center, Greenbelt, Maryland 20771.

[RUM] Rumbaugh, J., Blaha, M., et. al., *Object-Oriented Modeling and Design*, Prentice Hall, 1991.

[SHE] Shepperd, M., and Cartwright, M., "An Empirical Investigation of an Object-Oriented Software System", Department of Computing Bournemouth University Talbot Campus Poole, BH12 5BB UK, October 1997.

[TAN] Tang, M., Kao, M., and Chen, M., "An Empirical Study on Object-Oriented Metrics", *IEEE Transactions on Software Engineering,* 0-7695-0403-5, 1999.

[TEG] Tegarden, D., Sheetz, S., Monarchi, D., "Effectiveness of Traditional Software Metrics for Object-Oriented Systems", *Proceedings: 25th Hawaii International Confernce on System Sciences, January*, 1992, pp. 359-368.

[WEY] Weyuker, E., "Evaluating Software Complexity Measures", *IEEE Transactions on Software Engineering,* 14:1357-1365, 1988.