http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 5 (6): 1053-1057, 2006
ISSN 1812-5638

Analyzing the Software Quality Metrics for Object Oriented Technology

[1]S. Parthasarathy and [2]N. Anbazhagan
[1]Department of Computer Science and Engineering, [2]Department of Mathematics,
Thiagarajar College of Engineering, Madurai-625 015, India

Abstract: Metrics are units of measurement. Software engineering metrics are units of measurement that are used to characterize the software engineering products and software engineering processes. Object-oriented design and development is becoming very popular in today's software development environment. Object-Oriented Programming Structure (OOPS) requires not only a different approach to design and implementation; it requires a different approach to software metrics. Since object oriented technology uses objects and not algorithms as its fundamental building blocks, the approach to software metrics for object-oriented programs must be different from the standard metrics set. Object-oriented analysis and design focuses on objects as the primary agents involved in a computation. This study addresses the following questions: (i) what are the concepts in object-oriented technology that affect the software quality? (ii) How the various metrics found in the literature are useful to measure the critical concepts in object-oriented technology.

Key words: Metrics, object oriented design, algorithms

INTRODUCTION

Metrics are units of measurement. The term metrics is also frequently used to mean a set of specific measurements taken on a particular item or process. Software engineering metrics (Booch, 1994) are units of measurement that are used to characterize: (i) software engineering products e.g., designs, source code and test cases (ii) software engineering processes e.g., the activities of analysis, designing and coding and (iii) software engineering people e.g., the efficiency of an individual tester, or the productivity of an individual designer. If used properly, software engineering metrics allow us to (i) quantitatively define success and failure and/or the degree of success or failure for a product (ii) identify and quantify improvement, lack of improvement, or degradation in our products, processes (iii) make meaningful and useful managerial and technical decisions.

With object-oriented analysis and design methodologies gaining popularity, it is time to start investigating object-oriented metrics with respect to these goals. Object-oriented analysis and design focuses on objects as the primary agents involved in a computation. We are interested in addressing the following questions: (i) what are the concepts in object-oriented technology that affect the software quality? (ii) How the various metrics found in the literature are useful to measure the critical concepts in object-oriented technology.

DESIGN OF STUDY

Object-oriented design and development are popular concepts in today's software development environment; Object-oriented software development requires a different approach from more traditional functional decomposition and data flow development methods (Fenton, 1991; Lorenz and Jeff, 1994). Since the object-oriented metrics require a cursory understanding of the object-oriented concepts, we begin with an overview of basic object oriented structures. Then, we give an overview of the metrics for object-oriented systems. These metrics include modifications of traditional metrics as well as new metrics for specific object-oriented structures. We then discuss the object-oriented metrics with respect to their relationship to the attributes of software quality. After analyzing these metrics through empirical validations, we give a summary on how the various metrics found in the literature are useful to measure the critical concepts in object-oriented technology.

OOPS METRICS AND ITS APPLICATIONS

The metrics for object-oriented systems (Chidamber and Kemerer, 1994) are given below in two parts i.e., three traditional metrics and six new metrics specifically for object-oriented systems. The first three metrics in Table 1 are examples of how traditional metrics

Corresponding Author: S. Parthasarathy, Department of Computer Science and Engineering, Thiagarajar College of Engineering, Madurai-625 015, India

Table 1: Software metrics and OOPS applications

Type	Metric	OOPS application
Traditional	CC (Cyclomatic Complexity)	Method
Traditional	SIZE (Lines of code)	Method
Traditional	COM (Comment percentage)	Method
Object-oriented	WMC (Weighted Methods per Class)	Class/Method
Object-oriented	RFC (Response For a Class)	Class/Message
Object-oriented	LCOM (Lack of Cohesion of Methods)	Class/Cohesion
Object-oriented	CBO (Coupling Between Objects)	Coupling
Object-oriented	DIT (Depth of Inheritance Tree)	Inheritance
Object-oriented	NOC (No. of Children)	Inheritance

can be applied to the object-oriented structure of methods instead of functions or procedures. The next six metrics are specifically for object-oriented systems and the object-oriented construct applicable is indicated. Albeit software measurement is a key factor in managing, controlling and improving the software development process, software quality criteria are neither well defined nor easily measurable.

Cyclomatic Complexity (CC) is software metric that provides a quantitative measure of the logical complexity of a program. Line Count (LC) deals with counting all physical lines of code, the number of statements and the number of comment lines. The Comment Percentage (CP) is calculated by the total number of comments divided by the total lines of code less the number of blank lines. The Weighted Methods per Class (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 larger the number of methods in a class, the greater the potential impact on children since children will inherit all the methods defined in a class. The (Response for a Class) 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.

Lack of Cohesion of Methods (LCOM) measures the degree of similarity of methods by instance variable or attributes. High cohesion indicates good class subdivision. Coupling between Object Classes (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. The depth of a class within the inheritance (DIT) 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 Number Of Children (NOC) is the number of immediate subclasses subordinate to a class in the hierarchy. 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.

OOPS CONCEPTS AND SOFTWARE QUALITY

Object-oriented metrics must be able to focus on the combination of function and data as an integrated object. The object-oriented metric criteria are to be used to evaluate the following attributes of software quality: (i) Efficiency (ii) Complexity (iii) Understandability (iv) Reusability (v) Testability/Maintenance. The concept of inheritance helps us to introduce the reusability factor in our software development. It improves the software quality as it can be characterized as a powerful tool to control the complexity of the code needed to realize a system (Eliens, 2000).The encapsulation enhances software quality by ensuring safety via information hiding. An object has a public interface that objects can use to communicate with it. However, that same object can maintain private information and methods that can be changed at any time without affecting the other objects depend on it (Satzinger *et al*., 2002).

Encapsulation has two major impacts on metrics (i) The basic unit will no longer be the subprogram, but rather the object and (ii) we will have to modify our thinking on characterizing and estimating systems. The selected object-oriented metrics are primarily applied to the concepts of classes, coupling and inheritance. Information hiding plays a direct role in such metrics as object coupling and the degree of information hiding.

DATA COLLECTION

Data collected through various C++ programs are applied to the OOPS metrics specified (Chidamber and Kemerer, 1994) and the OOPS metric values provided by software testing tools and the values of metrics arrived at (Basili, 1996) is also utilised for analyting the software quality factors. The Software Assurance Technology

Table 2: Software metrics and its standard values

Metric	CC	LOC	CP	WMC	RFC	LCOM	CBO	DIT	NOC
Objective	Low	Low	20%-30%	Low	Low	Low	Low	Low	Low

Table 3: OOPS metrics and its calculated values obtained from C++ programs

Parameter	WMC	DIT	RFC	NOC	LCOM	CBO
Minimum	95.00	8.00	106.00	11.00	421.00	29.00
Maximum	1.00	0.00	0.00	0.00	0.00	0.00
Mean	13.41	1.30	33.19	0.16	8.23	5.91
Std. Dev	13.91	1.91	33.21	1.45	62.77	7.47

Center (SATC) at NASA Goddard Space Flight Centre gives a summary of the objectives for the values suggested in the above description of metrics as given in Table 2. The relation between important object oriented software quality concepts, quality metrics and object-oriented features identified by Rosenberg and Hyatt (1997) are shown in the Table 5. Table 5 will be useful for analysing the usage of OOPS metrics as quality factors based on the correlation analysis and the statistical data given in this study.

METHODOLOGY

In this study, we have used Regression and Correlation (Sharma, 1994) to find the relationship between the various software metrics for OOPS and its impact on software quality. The relationship between two variables such that a change in one variable result in a positive or negative change in the other and also greater change in one variable results in a corresponding greater change in the other is known as Correlation. If the second variable is unaffected by a change in the first, they are said to be statistically independent. We have used the Karl Pearson's Coefficient of correlation which is defined as follows:

$$R = \frac{1/N(\sum Xi * Yi - \overline{X} * \overline{Y})}{[1/N\sum Xi^2 - (\overline{X^2})]^{1/2}[1/N\sum Yi^2 - (\overline{Y^2})]^{1/2}}$$

Note that R has no units and is a mere number. If there exists some relation between two variables, their scatter diagram shall have points clustering near about some curve. If this curve is a straight line, it suggests some linear relationship between the variables and this straight line is known as the Line of Regression. The coefficient of regression of y on x is given by:

$$b_{YX} = \frac{covariance(x,y)}{Var(x)} = r\frac{\sigma Y}{\sigma X}$$

and similarly, the coefficient of regression of x on y is given by:

$$b_{XY} = \frac{covariance(x,y)}{Var(y)} = r\frac{\sigma X}{\sigma Y}$$

The regression lines are used to find the value of a dependent variable, generally, y for known value of x, the independent variable. In such a case, only one line of regression that of y on x is required. The independent variable is called predictor and the dependent variable the predictant. Correlation measures relationship but does not give the cause. The correlationship may be accidental or due to some third factor. However, if there is a cause, there is bound to be some sort of correlation. Regression gives a functional relationship between the variable X and Y, one being taken as the dependent and other the independent variable. It enables us to make predictions for the possible value of one variable for mutual variation and association between the variables, none being dependent or independent variable. Correlation is thus unsuitable for predictions.

DATA ANALYSIS

Table 3 and 4 provide common descriptive statistics of the metric distributions. These results indicate that inheritance hierarchies are somewhat flat (DIT) and that classes have, in general, few children (NOC). In addition, most classes show a lack of cohesion (LCOM) near zero. This latter metric does not seem to differentiate classes well and this may stem from its definition which prevents any negative measure. Table 4 shows very clearly that linear Pearson's correlations (R^2: Coefficient of determination) between the studied Object oriented metrics are, in general, very weak. Some of the R^2 values appear somewhat more significant. However, when applying the scatter diagram, only the relationship between CBO and RFC seems not to be due to outliers. We conclude that these metrics are mostly statistically independent and, therefore, do not capture a great deal of redundant information.

The values of R^2 can range between 0 and 1. When $R^2 = 1$, it means that all the points on the scatter diagram (Hooda, 2000) fall on the regression line and the entire variations are explained by the straight line. When $R^2 = 0$,

Table 4: Correlation analysis (R^2 values)

Metrics	WMC	DIT	RFC	NOC	LCOM	CBO
WMC	1.0	0.01	0.21	0	0.36	0.11
DIT	0.01	1.0	0	0	0.01	0
RFC	0.21	0	1.0	0	0.08	0.29
NOC	0	0	0	1.0	0	0
LCOM	0.36	0.01	0.08	0	1.0	0.01

Table 5: OOPS metrics and its relation with software quality factors

Metric	CC	LOC	CP	WMC	RFC	LCOM	CBO	DIT	NOC
OOPS feature	Method	Method	Method	Class/Method	Class/Method	Class/Cohesion	Coupling	Inheritance	Inheritance
Software quality factor	C	C	U, R	C, U, R	D, R, T	D, R	D, R	R, U, T	D

C-Complexity, U-Usability, R-Reusability, T-Testability, UR-Understandability, D-Design

it means none of the points on the scatter diagram falls on the regression line, meaning thereby that there is no relationship between the two variables. Here, R^2 provides the necessary link between regression and correlation which are the related aspects of a single problem of the analysis of the relationship between two variables. When $R^2 = 0$, there is no correlation between the two variables, When $R^2 = 1$, there is a perfect correlation. The values in Table 3 are helpful to identify the distribution as either positively skewed or negatively skewed (Hooda, 2000). The standard deviation values indicate the magnitude of deviation of the observations comprising a set of data in terms of their distance from the mean.

If the two lines of regression coincide, the correlation between the variables is perfect, the condition being

$$r\frac{\sigma Y}{\sigma X}=\frac{1\sigma Y}{r\sigma X}\ OR\ r^2=1\ \ OR\ \ r=\pm 1$$

If the variables x and y are independent, that is, the coefficient of correlation between them is zero, the lines of regression of y on x coincides with x-axis and that of x on y with y-axis and thus cut at right angles to each other. It is also easy to see that r, bxy and byx have the same sign since σ_X and σ_Y are always positive. Hence, from Table 4 we find that the lines of regression of various software metrics discussed earlier cut at right angles to each other.

DEFINING HYPOTHESIS AND FINDINGS

A number of empirical studies have been carried out for OOPS metrics. But we find that no explicit hypothesis is being evaluated. Two possible hypotheses with which to examine the empirical work shown in Table 3 and 4 are as follows:

- The existing OOPS metrics can be used to evaluate software complexity.
- The OOPS metrics with values as suggested in Table 2 and 5 is sufficient to ensure software quality.

As Table 3 and 4 indicates, the results of empirical validation of OOPS metrics do not give a great deal of support to either hypothesis. In general the results are not very compelling. Four classes of observations have been presented. They are: (1) It is clear that the ease of program comprehension is not completely orthogonal to software complexity. (2) The metric appears to be independent of generally accepted program structuring techniques. (3) As the various metrics are statistically independent, their ability to capture redundant information is null and void (4) Assessing the complexity of the software which will affect the software quality rigorously is not addressed.

As cyclomatic complexity is proved unsatisfactory on theoretical grounds by Martin (1988), it is necessary to ensure that existing OOPS metrics would account for software complexity too. As suggested by Rosenberg (1997) and as shown in Table 2 and 5, though the values of the various OOPS metrics are set to low, as more than one OOPS metrics accounts for a software quality factor and as these metrics are statistically independent and no existence of absolute correlation, the OOPS metrics (Chidamber and Kemerer, 1994) has to be refined and due care is still required to ensure that software quality can be measured and assured through the evaluation of the software via these metrics.

CONCLUSION AND FUTURE WORK

In this study, we discussed the software quality factors, software metrics for object oriented structures and the relation between important object oriented metrics and software quality factors. The work already carried out in this domain is also made use in this study and two hypotheses have been formulated. A set of objections have been presented towards the hypotheses and the statistical results are used to support our findings. Object oriented technology has many advantages, but many

unresolved issues need to be addressed before it can be fully utilized in the development of large-scale systems. It is evident that the software quality metrics for object-oriented technology plays a vital role in software projects and it seems that object-orientation will be mainstream software development approach of this decade. Work must be carried out to validate these metrics across different programming languages and platforms. Also we are working towards validating the other metrics mentioned in the literature and develop improved metrics. Many challenges still call for pragmatic solutions in order to make objects into productive blocks for large systems.

REFERENCES

Basili, V.R., 1996. Validation of object-oriented design metrics as quality indicators. IEEE Transactions on Software Engineering. Vol. 22, No. 10, October, 1996.

Booch, G., 1994. Object-oriented Analysis and Design with Applications. The Benjamin/Cummings Publishing Company.

Chidamber, S.R. and C .F. Kemerer, 1994. A metrics suite for object-oriented design. IEEE Transactions on Software Eng., 20: 476-493.

Eliens, A., 2000. Principles of Object Oriented Software Development. Addison Wesley, USA.

Fenton, N.E., 1991. Software metrics: A rigorous approach. Chapman and Hall.

Hooda, R.P., 2000. Statistics for Business and Economics. Macmillan India Ltd.

Lorenz, M. and K. Jeff, 1994. Object-Oriented Software Metrics. Prentice Hall Publishing.

Martin, S., 1988. A Critique of Cyclomatic Complexity as a Software Metric. Software Eng. J., 20: 30-36.

Rosenberg, L.H. and L.E. Hyatt, 1997. Software quality Metrics for Object-oriented Environments. NASA, SATC, http://satc.nasa.gov/support/CROSSAPR97/oocross. PDF.

Satzinger, J.W., R.B. Jackson and S.D. Burd, 2002. System Analysis and Design in a Changing World, 2nd Edn., Thomson Learning, Canada.

Sharma, G., 1994. Mathematical Statistics. KRISHNA Prakashan Mandir.