Object-Oriented Metrics that Predict Maintainability

Wei Li
Kollmorgen Industrial Drives, Radford, Virginia

Sallie Henry
Computer Science Department, Virginia Tech, Blacksburg, Virginia

Software metrics have been studied in the procedural paradigm as a quantitative means of assessing the software development process as well as the quality of software products. Several studies have validated that various metrics are useful indicators of maintenance effort in the procedural paradigm. However, software metrics have rarely been studied in the object-oriented paradigm. Very few metrics have been proposed to measure object-oriented systems, and the proposed ones have not been validated. This research concentrates on several object-oriented software metrics and the validation of these metrics with maintenance effort in two commercial systems. Statistical analyses of a prediction model incorporating 10 metrics were performed. In addition, a more compact model with fewer metrics is presented.

1. INTRODUCTION

Software engineering involves the study of the means of producing high-quality software products with predictable costs and schedules. One of the major goals in software engineering is to control the software development process, thereby controlling costs and schedules as well as the quality of the software products. As a direct result of software engineering research, software metrics have been brought to the attention of many software engineers and researchers. As DeMarco [1] pointed out, "you cannot control what you cannot measure." Software metrics provide a quantitative means to control the software development process and the quality of software products. However, effective use of software metrics depends on the statistical validation of the metrics.

Address correspondence to Prof. Sallie Henry, Computer Science Department, Virginia Tech, Blacksburg, VA 24061.

Software metrics have been studied in the procedural paradigm for more than a decade. Various software metrics have been proposed and studied, including Halstead's software science metrics [2], McCabe's cyclomatic metric [3], Henry and Kafura's information flow metric [4], Robillard's statement interconnection metric [5], Bail's HAC complexity [6], and Adamov's hybrid metrics [7]. These metrics were proposed primarily to measure software in the procedural paradigm.

Several attempts have been made to link software metrics with system maintainability in the procedural paradigm. Rombach [8, 9] indicated that software maintainability can be predicted using software metrics. Wake and Henry [10] showed that software maintainability can be predicted from software metrics. All of these preliminary results about the relationship of software metrics and system maintainability were obtained in the procedural paradigm. These results have yet to be verified in the object-oriented paradigm.

Object-oriented programming is another focus of the software engineering community. This paradigm claims a faster development pace and higher quality of software than the procedural paradigm. However, the use of metrics in the object-oriented paradigm has yet to be studied. So far, very few object-oriented metrics have been proposed [11], and the proposed metrics need to be validated. One possible way to validate metrics is to conduct statisical analyses of the metrics and measures of system maintainability.

Software maintenance is one of the most difficult and costly tasks in the software development process. Among all the factors that have a potential influence on software maintainability, software met-

655 Avenue of the Americas, New York, NY 10010
0164-1212/93/$6.00

rics, especially those that measure the interconnectivity of system components, have been shown to have an impact on software system maintainability in the procedural paradigm [9]. However, the relationship between metrics and system maintainability in the object-oriented paradigm is unclear. Furthermore, very little is known about how and where different maintenance activities are performed in the object-oriented paradigm.

This study attempts to bring the research in software metrics and the research in object-oriented programming together. Specifically, this study 1) investigates proposed object-oriented software metrics, 2) proposes some additional object oriented metrics, and 3) validates the metrics using the maintenance data collected from two commercial software systems.

2. SOFTWARE METRICS

Software metrics measure certain aspects of software. Software metrics generally can be divided into two categories; software product metrics and software process metrics. Software product metrics measure software products such as source code or design documents. Software process metrics measure the software development process, such as the number of man-hours charged to the development activities in the design and coding phases. This research focuses on software product metrics.

Software Metrics in the Procedural Paradigm

Several metrics have been proposed to measure the complexity of a procedure, a function, or a program in the procedural paradigm. These metrics range from simple size metrics such, as lines of code, to very complicated program structure metrics, such as Robillard's statement interconnectivity metrics. This section presents some sample metrics in the procedural paradigm.

Some of these metrics are lexical measures, which count certain lexical tokens in a program. These metrics include Halstead's software science metrics [2] and Bail's size metric [6]. Other measures are based on the analysis of patterns in a graph when the graph is constructed from the control flows of a program. One example is McCabe's cyclomatic metric [3]. McCabe defined the cyclomatic complexity measure based on the control flows in a procedure/function. A directed graph is derived based on the control flows of a procedure/function. The cyclomatic complexity is based on the complexity of the directed graph. For structural programs, an equivalent and simpler form of the cyclomatic complexity is the count of simple Boolean conditions in all control constructs (e.g., while, if, case, loop, etc.). McCabe later extended the cyclomatic complexity to measure structure chart design [12].

Another set of metrics measures the interconnection of system components. The interconnection may be based on the statements or components of a program, such as procedures or functions. Some examples include McClure's invocation metric [13], Henry and Kafura's information flow metric [4], Woodfield's review metric [14], Adamov's hybrid metric [7], and Robillard's statement interconnection metric [5].

The pioneering work of defining software metrics has concentrated on measures of complexity in the procedural paradigm. Since the object-oriented paradigm exhibits different characteristics from the procedural paradigm, software metrics in the object-oriented paradigm need to be studied. The difference between the procedural paradigm and the object-oriented paradigm is due to the difference between the programming philosophies in the two paradigms. For example, the object-oriented concepts of inheritance, classes, and message passing cannot be characterized by any of the metrics mentioned above.

The research of software metrics in the procedural paradigm lacks justification for program behaviors [15]. This research studies the basic concepts of the object-oriented paradigm and associated programming behaviors before proposing any metrics.

Software Metrics in the Object-Oriented Paradigm

Understanding the object-oriented paradigm is the first step toward defining metrics for that paradigm. The study of the object-oriented paradigm results in object-oriented concepts such as object, class, attributes, inheritance, method, and message passing.

The programming behaviors exhibited in the object-oriented paradigm differ from those of the procedural paradigm. For example, the creation of classes in the object-oriented paradigm is a programming behavior distinguished from the creation of procedure/functions in the procedural paradigm. Each basic object-oriented concept implies a programming behavior. Therefore, each basic concept is studied and a metric proposed for that particular concept.

The second step in defining metrics in the object-oriented paradigm is to establish a theoretical foundation for the metrics. Chidamber and Kemerer [11]

present a measurement theory base for measuring complexity in the object-oriented paradigm. Proposed object-oriented metrics are not as numerous as those in the procedural paradigm. Chidamber and Kemerer propose a suite of six object-oriented design metrics: depth of the inheritance tree (DIT), number of children (NOC), coupling between objects (CBO), response for a class (RFC), lack of cohesion of methods (LCOM), and weighted method per class (WMC) [11]. Coupling between objects was not previously defined. Three coupling metrics are presented which attempt to focus on different aspects of coupling. Finally, two size metrics are presented.

3. OBJECTED-ORIENTED METRICS DEFINITIONS

Three groups of object-oriented metrics are investigated here. The first group contains all the metrics proposed by Chidamber and Kemerer [11]. Some additional metrics are discussed in the second group. The last group discusses some size metrics in the object-oriented paradigm.

Five Proposed Object-Oriented Metrics

We used five of the six metrics proposed by Chidamber and Kemerer [11]: DIT, NOC, RFC, LCOM, and WMC. We did not use CBO, which measures "noninheritance-related coupling" [11].

The DIT metric measures the position of a class in the inheritance hierarchy [11]. It addresses the inheritance concept discussed earlier. It seems logical that the lower a class is in the inheritance tree, the more superclass properties this class may access because of its inheritance. If the subclass accesses the inherited properties from the superclass without using the methods defined in the superclass, the encapsulation of the superclass is violated. One may hypothesize that the larger the DIT metric, the harder it is to maintain the class. The calculation of the DIT metric is the level number for a class in the inheritance hierarchy. The root class's DIT is zero:

DIT = inheritance level number

ranging from 0 to N; where N is positive integer.

The NOC metric measures the number of direct children a class has [11]. This metric addresses the inheritance concept mentioned earlier, but from a different perspective than DIT. It seems logical that the more direct children a class has, the more classes it may potentially affect because of inheritance. For example, if there are many subclasses of the class that are dependent on some methods or instance variables defined in the superclass, any changes to these methods or variables may affect the subclasses. One may intuit that the larger the NOC metric, the harder it is to maintain the class. The calculation of NOC is as follows:

NOC = number of direct sub-classes

ranging from 0 to N; where N is a positive integer.

The RFC metric measures the cardinality of the response set of a class. The response set of a class consists of all local methods and all the methods called by local methods [11]. The RFC metric addresses the method concept discussed earlier. It seems logical that the larger the response set for a class, the more complex the class. One may intuit that the larger the RFC metric, the harder it is to maintain the class, because calling a large number of methods in response to a message makes tracing an error difficult. The calculation of RFC is:

RFC = number of local methods
+ number of methods called by local methods

ranging from 0 to N; where N is a positive integer.

The LCOM metric measures the lack of cohesion of a class [11]. The cohesion of a class is characterized by how closely the local methods are related to the local instance variables in the class. This metric addresses the class and method concepts discussed earlier. It seems logical that the more cohesive a class, the easier the class is to maintain. One may intuit that the larger the metric, the harder it is to maintain the class, because, if all the methods defined in a class access many independent sets of data structures encapsulated in the class, the class may not be well designed and partitioned. The calculation of LCOM is the number of disjoint sets of local methods. Disjoint sets are a collection of sets that do not intersect with each other. Any two methods in one disjoint set access at least one common local instance variable [11]:

LCOM = number of disjoint sets of local methods

No two sets intersect and any two methods in the same set share at least one local instance variable ranging from 0 to N; where N is a positive integer.

The WMC metric measures the static complexity of all the methods [11]. This metric addresses the class and method concepts discussed above. It seems logical that the more methods, the more complex the class. The more control flows a class's methods have, the harder it is to understand them, thus, the harder it is to maintain them. The WMC is calcu-

lated as the sum of McCabe's cyclomatic complexity of each local method:

WMC = summation of McCabe's cyclomatic complexity of all local methods

ranging from 0 to N; where N is a positive integer.

Definition of Additional Object-Oriented Metrics

Two objects are coupled if they act on each other [11]. Three types of object coupling are identified in this research. The objects are coupled through certain communication mechanisms provided by the object-oriented paradigm. The forms of object coupling are coupling through inheritance, coupling through message passing, and coupling through data abstraction.

Coupling through inheritance. Inheritance promotes software reuse in the object-oriented paradigm. However, it also creates the possibility of violating encapsulation and information hiding. This violation occurs because the properties in the superclass are exposed to the subclass for less restrictive access. The use of inheritance that is not well designed may introduce extra complexity to a system. The extra complexity is due to the attributes, which are encapsulated in the superclass but now are exposed to less restrictive accesses by the subclass. The more a class inherits, the more nonprivate attributes the class may access.

DIT and NOC are used to measure the inheritance characterization. DIT indicates how many superclasses a class has, thus, it indicates how many classes the class is dependent on. NOC indicates how many classes may be directly affected by the class. Both DIT and NOC metrics were discussed and defined earlier.

Coupling through message passing. One communication channel the object-oriented paradigm allows is message passing. When an object needs some service that other objects provide, messages are sent from that object to the other objects. A message is usually composed of the object ID, the service (method) requested, and the parameter list for the method. Although messages are passed among objects, the types of messages passed are defined in classes. Therefore, message passing is calculated at the class level instead of the object level.

Message-passing coupling (MPC) is used to measure the complexity of message passing among classes in this research. Since the pattern of the message is defined by a class and used by objects of the class, the MPC metric also gives an indication of how many messages are passed among objects of the classes:

MPC = number of send statements defined in a class.

The number of messages sent out from a class may indicate how dependent the implementation of the local methods is on the methods in other classes. This may not be indicative of the number of messages received by the class.

Coupling through abstract data types. The concept of an abstract data type (ADT) is discussed in Korson and McGregor [16]. A class can be viewed as an implementation of an ADT [16]. A variable declared within a class X may have a type of ADT which is another class definition, thereby causing a particular type of coupling between X and the other class because X can access the properties of the ADT class. This type of coupling may cause a violation of encapsulation if the programming language permits direct access to the private properties in the ADT. The metric that measures the coupling complexity caused by ADTs is data abstraction coupling (DAC):

DAC = number of ADTs defined in a class.

The number of variables having an ADT type may indicate the number of data structures dependent on the definitions of other classes. The more ADTs a class has, the more complex the coupling of that class with other classes.

A Class Interface Increment Metric

Another metric used in this research is the number of methods (NOM) in a class. Because the local methods in a class constitute the interface increment of the class, NOM serves best as an interface metric. NOM is easy to collect in most object-oriented programming languages.

NOM = number of local methods.

The number of local methods defined in a class may indicate the operation property of a class. The more methods a class has, the more complex the class's interface.

Two Size Metrics

Size has been used as a software metric for a long time. The lines of code (LOC) metric is used to measure a procedure or a function and the accumu-

lated LOC of all procedures and functions is used to measure a program. However, the size factor in an object-oriented program has not been well established.

Besides the metrics discussed above, two size metrics are also used in this research. One is the traditional LOC metric, which is calculated by counting the number of semicolons in a class. The LOC metric is hereafter referred to as SIZE1:

SIZE1 = number of semicolons in a class.

The second size metric used in this research is the number of properties (including the attributes and methods) defined in a class. This size metric is referred to as SIZE2 and is calculated as follows:

SIZE2 = number of attributes
+ number of local methods.

4. TOOLS AND DATA

Classic-Ada™ is an object-oriented design/programming language developed by Software Productivity Solutions, Inc. The two commercial software products, UIMS™ (User Interface System) and QUES™ (QUality Evaluation System), used in this research were designed and developed with Classic-Ada.[1] A Classic-Ada metric analyzer was constructed to collect the metrics from the Classic-Ada design and source code. The analyzer was implemented using LEX and YACC in UNIX™ environments.[2] Maintenance effort data have been collected over the past 3 years from the SPS™ environment.

Classic-Ada is used as the object-oriented design/programming language in this research. The two commercial systems analyzed in this research were designed and implemented in Classic-Ada by SPS. A Classic-Ada metric analyzer was designed and implemented on the Mach operating system running on a NeXTstation using a GNU C compiler. The system was ported to an Ultrix system running on a VAX station. The analyzer uses LEX and YACC utilities with C as their embedded language.

Classic-Ada Design /Programming Language

Classic-Ada is an object-oriented design/programming language. It brings the capability of object-oriented programming to Ada by adding object-oriented constructs to the Ada constructs. It supports all the standard constructs defined in ANSI/MIL-STD-1815A. In addition to the standard Ada constructs, Classic-Ada supports the object-oriented features with nine new constructs: class, method, instance, superclass, send, self, super, instantiate, and destroy. Classic-Ada supports data encapsulation, information hiding, and inheritance.

Classic-Ada Metric Analyzer

A metric analyzer was constructed to collect metrics from the Classic-Ada designs and source code. The metrics analyzer contains two passes. The first pass parses Classic-Ada definitions and generates an intermediate language file. The second pass parses the intermediate language file and calculates the metrics.

Maintenance Data

The maintenance effort data have been collected from two commercial systems designed and implemented using Classic-Ada. The two systems are UIMS and QUES. The data have been collected over the past 3 years. The maintenance effort is measured by the number of lines changed per class. A line change could be an addition or a deletion. A change of the content of a line is counted as a deletion and an addition. This measurement is used in this study to estimate the maintainability of the object-oriented systems.

The metrics discussed in this paper are abbreviated as follows:

DIT, depth in the inheritance tree

NOC, number of children

MPC, message-passing coupling

RFC, response for class

LCOM, lack of cohesion of methods

DAC, data abstraction coupling (cls_object_d)

WMC, weighted method complexity

NOM, number of methods

SIZE1, number of semicolons per class

SIZE2, number of methods plus number of attributes

The maintenance effort (collected for each class maintained) used in the study is

change = number of lines changed per class in its maintenance history

[1] SPS, Classic-Ada, UIMS, and QUES are trademarks of Software Productivity Solutions, Inc.

[2] UNIX is a trademark of Bell Laboratory.

The Appendix shows the data collected from UIMS and QUES systems. For a more detailed discussion of the data and the two systems, see Li [17].

5. THE RELATIONSHIP BETWEEN METRICS AND MAINTENANCE EFFORT

Rombach [8, 9] has suggested that, in the procedural paradigm, software maintainability correlates strongly with software metrics. However, knowledge of the relationship between software maintainability and software metrics in the object-oriented paradigm is sparse. The goals of the statistical analyses are to identify any relationship between metrics and maintenance effort and find a compact model that performs essentially as adequate a job as the full model.

Some terms used in this article require definitions:

Full model: a regression model containing all of the metrics discussed; the independent variables are DIT, NOC, MPC, LCOM, RFC, DAC, WMC, NOM, SIZE1, and SIZE2.

Size model: a regression model containing only the SIZE1 and SIZE2 metrics.

Sample: a set of classes drawn from a commercial environment (SPS).

Population: the set of all the classes existing in a commercial environment (SPS).

Dependent variable: a random variable with value to be predicted.

Independent variable: a predictor variable.

R-square: the percentage of the variance in the dependent variable accounted for by the independent variables in a regression model based on the sample data.

Adjusted R-square: the percentage of the variance in the dependent variable accounted for by the independent variables in a regression model in the population.

Sum of squares of regression: the variability of the dependent variable explained by the regression model.

Sum of squares of residual: the unexplained variability in the dependent variable.

Mean squares of residual: the sum of squares of the residual divided by the degrees of freedom of the residual.

Two main hypotheses are tested in the statistical analyses. The first hypothesis states that there is a strong relationship between object-oriented metrics and the maintenance effort as measured. This hypothesis is tested in the preliminary analyses. The second hypothesis states that there is redundancy among all the metrics used in the preliminary analyses section. The second hypothesis is tested in the refined analyses.

Preliminary Analyses

The results of the preliminary analyses are presented in Analysis 1, Analysis 2, and Analysis 3. These analyses are designed to determine if the maintenance effort can be predicted from metrics and determine if the size metrics are the sole major predictors. The maintenance effort, change, is measured as "the number of lines changed per class." Change is used as a dependent variable in this study.

Analysis 1 shows the regression analysis using change as the dependent variable and all the metrics discussed in this study as the independent variables. The high R-square (0.9096 in UIMS and 0.8773 in QUES) and adjusted R-square (0.8773 in UIMS and 0.8550 in QUES) and the high significance level (0.0001 in both UIMS and QUES) show with high confidence that most of the variance in the dependent variable "change" is accounted for by the metrics used in the test. The analysis concludes that the prediction of the maintenance effort as measured by change is possible from the metrics.

Analysis 2 shows the regression analysis using change as the dependent variable and the two size metrics as the independent variables. Along with Analysis 3, this test is designed to determine the effect of size metrics in the regression analysis. The R-square (0.6617 in UIMS and 0.6282 in QUES) and adjusted R-square (0.6429 in UIMS and 0.6172 in QUES) demonstrate with high confidence (0.0001 significance level in both UIMS and QUES) that a large portion of the variance in the maintenance effort can be predicted from the size metrics. Analysis 2 concludes that the size metrics are important predictors.

To determine scientifically if size metrics are the sole major predictors in the regression model, Analysis 3 is used to test the null hypothesis (H0), which states that "the size model is not essentially as good as the full model," or "there is no difference between the full model and the size model." Analysis 3 shows the result of a partial F test to decide if the size model is essentially as good as the full model. The rejection of the null hypothesis at 0.005 signifi-

cance level in UIMS and 0.001 in QUES shows that the size metrics are not the sole predictors in the models. The analysis concludes that the metrics (DIT, NOC, MPC, RFC, LCOM, DAC, WMC, NOM) contribute to the prediction of the maintenance effort above and beyond what can be predicted from the size metrics alone.

Analysis 1: The full model regression of maintenance effort in UIMS and QUES. This analysis is designed to answer the question, "Are there object-oriented metrics that can predict maintenance effort?"

Dependent variable = change

Independent variables = SIZE1 + SIZE2 + DIT + NOC + MPC + RFC + LCOM + DAT + WM + NOM

Probability > F = 0.0001 in UIMS (0.0001 in QUES)

R-square = 0.9096 in UIMS (0.8737 in QUES)

Adjusted R-square = 0.8773 in UIMS (0.8550 in QUES)

MS.Residual = 462.59501 in UIMS (59.51347 in QUES)

There are 1 dependent and 10 independent variables in the full model. The dependent variable "change" is a measure of maintenance effort. All the independent variables are metric values.

Probability > F, commonly known as the p value, indicates the significance of a regression. The smaller the probability, the more significant the regression. Probability > F in both the UIMS and QUES regressions are at the 0.0001 level. This small value indicates, with a high degree of confidence, that some prediction is possible.

R-square is a regression quality indicator that measures the quality of predictions; that is, how much variance in the dependent variable is accounted for by the independent variables in the sample. Adjusted R-square measures the same aspect as R-square, but in the population, with the adjustment depending on both sample size and the number of independent variables.

In the UIMS system, > 90% of the total variance in the maintenance effort is accounted for by the metrics in the sample, and > 87% in the population. In the QUES system, > 87% of the variance in the maintenance effort is accounted for by the metrics in the sample and > 85% in the population.

We conclude that the prediction of maintenance effort from metrics is possible.

Analysis 2: The size model regression of maintenance effort in UIMS and QUES. This analysis is a supplement to Analysis 1. It is designed to examine the effect of the size metrics in predicting the maintenance effort.

Dependent variable = change

Independent variables = SIZE1 + SIZE2

Probability > F = 0.0001 in UIMS (0.0001 in QUES)

R-square = 0.6617 in UIMS (0.6282 in QUES)

Adjusted R-square = 0.6429 in UIMS (0.6172 in QUES)

MS.Residual = 1346.14155 in UIMS (157.16593 in QUES)

There are one dependent and two independent variables. The dependent variable, change, is the same as in Analysis 1. The purpose of this analysis is to determine if size metrics alone can predict maintenance effort. The high significance level (probability > F = 0.0001 in both UIMS and QUES) displays a high degree of confidence that this prediction is possible. The R-square values (0.6617 in UIMS and 0.6282 in QUES) demonstrate that a significant portion of the variance in the maintenance effort is accounted for by the size metrics.

We conclude that size metrics can account for much of the total variance in maintenance effort.

Analysis 3: Comparison of full models with the size model. This analysis is designed to determine if size metrics are the only major predictors in the full model. The null hypothesis (H0) and the alternative hypothesis (H1) are as follows:

H0: there is no difference between the full model and the size model.

H1: there is a difference between the full model and the size model.

Partial F (observed) = 9.594898351 (16.9396491 in QUES)

n = 39 (71 in QUES)

F (critical) in UIMS = $F(8, 28) = 6.5\alpha = 0.005$

F (critical) in QUES = $F(8, 60) = 9.92\alpha = 0.001$

H0 is rejected because F (observed) > F(critical) at α = 0.005 in UIMS and α = 0.001 in QUES.

Partial F tests are formed for comparing the full model with the size model for UIMS and QUES. Because the observed F values (9.59 in UIMS and 16.94 in QUES) are greater than the critical F

values (6.5 in QUES and 9.92 in QUES), the null hypothesis (H0) is rejected at the 0.005 significance level in UIMS and 0.001 in QUES. The rejection of the null hypothesis means that size metrics are not the sole major predictors in the full model. Thus, the other metrics (DIT, NOC, MPC, RFC, LCOM, DAC, WMC, and NOM) contribute to the prediction of the maintenance effort above and beyond what can be predicted using the size metrics alone.

We conclude that the metrics are useful predictors of maintenance effort.

Refined Analyses

This section discusses the refined regression analyses of the maintenance effort using fewer metrics than the preliminary analyses. The preliminary analyses concluded that prediction of maintenace effort is possible. However, not all the independent variables used in the previous analyses are necessary. The criterion used to eliminate redundant predictors is the variation inflation factor (VIF) of each predictor. If the VIF for a predictor is too high, then the predictor should be eliminated from the prediction equation. The criterion for the final set of metrics to be used in the analyses is that no one metric has a VIF value > 50. Table 1 gives the VIF for each predictor used in Analysis 1.

Table 1 shows that SIZE2 and NOM have very high VIF values (> 100). But this does not mean that both of them should be eliminated, because the elimination of one predictor would affect the VIFs for the remaining predictors. Some other factors considered in the elimination of the predictors are the ease of collecting the metrics and previous research results showing the correlations of some metrics in the procedural paradigm. For example, a high correlation of McCabe's complexity and several other metrics is found in Wake and Henry [10]. NOM is easier to collect than SIZE2; therefore, SIZE2 is eliminated. The traditional size metric SIZE1 correlates highly with McCabe's metric (WMC); therefore, SIZE1 is eliminated. The final prediction model contains fewer predictors. The predictors in the final compact code model and their VIF values are listed in Table 2. The VIF for RFC is reduced to a tolerable level by the deletion of SIZE1 and SIZE2 because RFC was relatively strongly correlated with SIZE1 and SIZE2.

Table 1. Variation Inflation Factor for the Independent Variables

Variable	VIF	
	UIMS	QUES
SIZE1	40.1969	11.4987
SIZE2	946.8073	380.6639
DIT	2.0607	1.9530
NOC	1.5209	not available
MPC	8.7454	4.0227
RFC	54.5136	14.0011
LCOM	4.0603	13.8002
DAC	77.3646	30.7386
WMC	38.1128	8.7724
NOM	599.0342	241.4471

Table 2. Variable Inflation Factors for the Final Independent Variables

Variable	VIF	
	UIMS	QUES
DIT	1.6524	1.8642
NOC	1.5093	not available
MPC	5.2090	2.7611
RFC	33.0560	11.3295
LCOM	3.2991	12.1277
DAC	5.2037	7.4529
WMC	7.7585	3.1365
NOM	27.8300	21.3158

Because no predictor has a VIF value > 50, all of the predictors shown in Table 2 are used in the refined regression analyses. Therefore, a compact model has been identified and the results of the refined regression analyses of the compact model are shown in Analysis 4. Analysis 5 shows the bidirectional cross-validations of the prediction equations obtained from Analysis 4. Analysis 4 is designed to determine if the compact model of metrics predicts maintenance effort. Analysis 5 is designed to cross-validate the results obtained from Analysis 4.

The small p values ($p = 0.0001$ for both UIMS and QUES) confirm, with a high degree of confidence, that maintenance effort can be predicted from this compact model. The high R-square values (0.9030 in UIMS and 0.8680 in QUES) and adjusted R-square (0.8771 in UIMS and 0.8533 in QUES) show that the quality of the prediction is quite reliable.

The prediction equation for UIMS was cross-validated by using it to predict the maintenance effort on the QUES data; the prediction equation derived from QUES was checked by applying it to the UIMS data. In each case, the correlation between predicted and actual maintenance effort was computed. Relatively strong correlations (0.65082 in UIMS and 0.6782 in QUES) confirm that predictions are reasonably accurate. Because a positive correla-

tion is expected, a one-sided t-test is performed. This test indicates that the prediction equations are valid throughout the population with a high degree of confidence.

Analysis 4: The compact model regression of maintenance effort in UIMS and QUES. This analysis is designed to answer the question, "Are there object-oriented metrics that can predict maintenance effort?" but using a more compact model than in Analysis 1.

Dependent variable = change

Independent variables = DIT + NOC + MPC + RFC + LCOM + DAC + WMC + NOM

Probability $> F = 0.0001$ in UIMS (0.0001 in QUES)

R-square = 0.9030 in UIMS (0.8680 in QUES)

Adjusted R-square = 0.8771 in UIMS (0.8533 in QUES)

There are one dependent and eight independent variables in the model. The dependent variable, change, is the measure of maintenance effort. The small p values (p = 0.0001 for both UIMS and QUES) confirm, with a high degree of confidence, that maintenance effort can be predicted from this compact model. The high R-square values (0.9030 in UIMS and 0.8680 in QUES) and adjusted R-square (0.8771 in UIMS and 0.8533 in QUES) show that the quality of the prediction is quite reliable.

We conclude that the prediction of maintenance effort is possible from the compact code model.

Analysis 5: Cross-validation of compact prediction models. This analysis is a supplement to Analysis 4. It is designed to determine if the conclusion from Analysis 4 is valid in the entire SPS software development environment.

Prediction Models

	Coefficient	
Predictor	UIMS	QUES
Intercept	−2.20372	6.808372
DIT	2.495030	−2.151131
NOC	5.368791	0.0
MPC	−2.58682	0.169582
RFC	1.797583	−0.14156
LCOM	2.762436	−2.195476
DAC	12.92241	−0.950382
WMC	2.523366	0.886097
NOM	−6.78521	1.94425

The Null hypothesis (H0) and the alternative hypothesis (H1) are as follows:

H0: there is no relationship between the predicted change and the observed change.
H1: there is a positive relationship between the predicted change and the observed change.

r = 0.65082 in UIMS (0.6782 in QUES)

$t.obs$ = 7.12051 in UIMS (5.61363 in QUES)

$t.crit$ = 3.232 (α = 0.001) in UIMS

$t.crit$ = 3.385 (α = 0.001) in QUES

H0 rejected at α = 0.001 in both UIMS and QUES.

The correlation of the predicted change and the observed change is represented by r. An r value of 0.65082 in UIMS and 0.6782 in QUES represents reasonably high correlations for cross-validation. The number of observations in the sample is indicated by n. The t values are represented by $t.obs$ for the observed t and $t.crit$ for the critical t. The significance level of a cross-validation is indicated by an α value. A commonly accepted α value is 0.05. An α value of 0.001 in both cross-validation shows a high degree of confidence for the successful validations.

We conclude that the compact code model prediction equation is valid in the population.

6. CONCLUSIONS

The aim of this research was to

1. Implement the proposed metrics for the object-oriented paradigm
2. Propose and implement additional metrics for the object-oriented paradigm
3. Investigate these metrics and their relationship with the maintenance effort
4. Derive a prediction model for the maintenance effort measure using object-oriented metrics
5. Validate the model on two object-oriented systems.

Various statistical analysis procedures were used. Multiple linear regression was the main statistical tool used.

The results of the analyses of the two object-oriented systems show that

1. There is a strong relationship between metrics and maintenance effort in object-oriented systems.
2. Maintenance effort can be predicted from combinations of metrics collected from source code.
3. The prediction is successfully cross-validated.

APPENDIX

UIMS System Data

Class	DIT	NOC	MPC	RFC	LCOM	DAC	WMC	NOM	SIZE2	SIZE1	Change
action	1	3	4	14	5	1	3	7	9	37	14
alert	3	0	6	20	6	2	5	7	9	94	18
boolean_data	2	0	1	12	4	1	1	6	7	26	2
cancel	2	0	1	2	1	0	1	1	1	4	2
context	1	0	1	21	12	8	0	18	26	60	10
data	1	8	1	7	3	1	0	4	5	12	16
dialog	3	1	2	12	6	1	0	6	7	26	16
dictionary	2	0	1	10	7	1	9	7	8	59	18
done	2	0	1	2	1	0	1	1	1	4	2
enumeration_rendering	3	0	9	24	5	1	5	5	6	81	16
float_data	2	0	1	12	4	1	1	6	7	26	2
float_rendering	3	0	3	17	6	3	3	12	15	74	48
graphic	1	4	10	67	6	21	23	40	61	194	205
horizontal_view	3	0	9	30	9	0	16	9	9	143	30
indented_list	3	0	4	12	6	0	2	6	6	79	30
integer_data	2	0	1	12	4	1	1	6	7	26	2
integer_rendering	3	0	3	9	5	0	3	5	5	33	12
line	2	0	3	17	9	2	12	9	11	83	50
list_data	2	0	3	46	26	2	30	31	33	283	26
menu_title	3	0	7	29	7	3	8	13	16	90	39
name_dialog	4	0	4	10	4	0	0	4	6	61	15
object	0	6	1	12	8	3	8	10	13	71	289
offset_ratio	3	0	1	12	4	1	1	6	7	34	2
popup_menu	3	0	2	9	4	0	0	4	4	26	18
popup_window	2	4	6	29	6	2	9	11	13	86	26
quit	2	0	1	2	1	0	1	1	1	6	2
ratio	2	1	1	15	5	2	3	9	11	44	2
rectangle	2	0	3	11	5	0	7	5	5	45	48
selector	3	0	9	25	7	3	8	10	14	131	34
screen	1	0	11	46	13	5	37	20	26	296	93
string80_data	2	0	1	11	5	1	1	7	8	27	2
string80_rendering	2	6	5	40	13	6	44	20	26	316	168
tf_boolean_rendering	3	0	6	17	6	0	10	6	6	76	30
tree	2	0	4	54	6	2	45	32	34	338	17
uims	0	0	6	32	7	2	10	14	18	84	27
vertical_view	3	0	9	30	9	0	16	9	9	142	30
view	2	3	12	101	31	5	69	39	44	439	253
window	1	1	10	57	20	13	41	32	45	419	192
yn_boolean_rendering	3	0	6	17	6	0	10	6	6	76	20

QUES System Data

Class	DIT	NOC	MPC	RFC	LCOM	DAC	WMC	NOM	SIZE2	SIZE1	Change
core_list_data	0	0	2	27	5	1	38	24	25	365	102
data_select_window	2	0	18	39	3	2	4	5	9	174	85
description_window	2	0	11	29	4	1	2	5	7	122	38
equation_data	2	0	20	96	26	5	28	31	36	485	81
equation_node_data	2	0	8	62	10	8	31	29	37	310	55
fw_abstract_data	2	0	27	150	19	12	60	35	49	648	101
fw_abstract_node_data	2	0	14	68	18	5	14	25	31	260	38
fw_abstract_window	2	0	38	59	4	1	2	5	7	365	157
framework_definition_data	2	0	18	86	14	8	23	23	32	376	68
fw_definition_window	2	0	24	40	4	2	2	5	8	144	26
fw_level_data	2	0	8	54	14	4	13	19	24	211	24
fw_level_select_window	2	0	18	38	4	1	2	5	7	230	86
fw_phase_action	1	0	10	22	4	0	7	4	4	121	26
fw_phase_data	2	0	13	68	14	3	17	18	23	245	47
fw_phase_window	2	0	16	32	4	1	2	5	9	174	78

QUES System Data (*Continued*)

Class	DIT	NOC	MPC	RFC	LCOM	DAC	WMC	NOM	SIZE2	SIZE1	Change
fw_specific_data	2	0	23	110	18	14	48	38	53	534	88
fw_specific_window	2	0	20	41	5	1	2	6	10	250	124
gio_action	1	0	12	22	4	0	13	4	4	172	28
gio_definition_selector	2	0	13	22	3	1	2	4	8	142	62
gio_definition_window	2	0	16	42	5	3	9	6	10	210	35
gio_view_action	1	0	17	38	6	0	11	6	6	168	41
gio_view_window	2	0	32	55	5	4	7	6	11	259	49
list_looker	3	0	5	24	11	5	3	11	16	128	9
main_action	2	0	20	38	3	1	33	4	7	499	70
main_window	2	0	41	50	4	2	1	5	7	162	46
personnel_window	2	0	29	38	3	1	2	4	6	181	42
proj_abstract_data	2	0	21	149	23	9	41	35	46	613	92
proj_abstract_window	2	0	23	41	4	1	3	5	7	184	48
project_data	2	0	21	95	15	7	22	24	33	356	56
proj_definition_action	1	0	38	90	7	0	83	7	8	854	217
proj_definition_window	2	0	42	58	4	2	3	5	8	238	45
proj_level_data	2	0	8	54	14	4	13	19	24	211	24
proj_level_window	2	0	19	40	3	2	8	5	9	200	85
proj_role_data	2	0	8	34	12	3	9	15	19	146	10
proj_roles_window	2	0	21	42	4	1	5	5	7	233	100
proj_specific_data	2	0	17	118	24	8	32	35	44	505	72
proj_specific_window	2	0	22	40	4	1	3	5	7	183	48
property_dialog	4	0	17	37	6	9	10	6	15	188	24
ques_data	2	0	8	35	8	4	5	13	18	149	16
ques_persistent_string80_data	1	0	6	35	14	3	17	20	24	176	14
role_select_window	2	0	16	35	4	1	2	5	7	183	82
report_constraint_window	2	0	16	39	5	3	12	6	10	204	39
report_definition_action	1	0	16	43	7	0	40	7	7	354	98
report_definition_window	1	0	33	62	6	3	15	7	11	349	56
report_field	3	0	21	109	16	25	62	57	82	1009	146
report_field_rendering	3	0	12	37	5	1	26	8	9	236	25
report_selector	2	0	17	30	4	1	2	5	9	164	68
report_view_window	1	0	23	48	5	3	9	6	10	199	48
son_of_abstract_data	2	0	18	156	29	6	46	37	45	558	170
son_of_abstract_node_data	2	0	27	125	24	5	35	32	38	540	80
son_of_abstract_window	2	0	21	41	4	1	2	5	7	252	148
son_of_definition_window	2	0	18	34	4	2	2	5	8	135	30
son_of_equation_data	2	0	11	47	15	3	10	19	22	177	28
son_of_equation_node_data	2	0	9	56	20	4	12	26	30	206	35
son_of_level_select_window	2	0	15	34	4	1	2	5	7	217	77
son_of_phase_data	2	0	11	88	20	3	22	25	30	314	45
son_of_phase_window	2	0	15	31	4	1	2	5	9	153	52
son_of_specific_data	2	0	23	121	33	5	36	40	46	481	70
son_of_specific_window	2	0	24	45	5	1	3	6	10	327	188
template_abstract_window	2	0	20	39	4	1	2	5	7	216	79
template_definition_window	2	0	18	34	4	2	2	5	8	135	30
template_level_select_window	2	0	16	35	4	1	2	5	7	216	75
template_phase_window	2	0	15	32	4	1	2	5	9	153	64
template_specific_window	2	0	20	39	5	1	2	6	9	258	107
tool_data	2	0	6	33	12	4	7	15	20	134	8
tool_element	2	0	5	30	12	3	6	14	18	115	6
tools_action	1	0	14	27	3	2	14	4	7	194	24
tools_window	2	0	10	17	3	3	1	4	11	115	52
user_data	2	0	18	78	14	9	22	21	31	333	38
user_role_data	2	0	11	62	13	7	27	26	34	285	41
user_roles_window	2	0	17	40	4	1	5	5	7	183	94

REFERENCES

1. T. DeMarco, *Controlling Software Projects: Management, Measurement and Estimation*, Yourdon Press, New Jersey, 1982.
2. M. H. Halstead, *Elements of Software Science*, Elsevier North-Holland, New York, 1977.
3. T. J. McCabe, A Complexity Measure, *IEEE Trans. Software Eng.* 2, 308–320 (1976).
4. S. Henry and D. Kafura, Software Structure Metrics Based on Information Flow, *IEEE Trans. Software Eng.* 7, 510–518 (1981).
5. P. Robillard and G. Boloix, The Interconnectivity

Metrics: A New Metric Showing How a Program is Organized, *J. Syst. Software* 10, 29–39 (1989).
6. W. G. Bail and M. V. Zelkowitz, Program Complexity Using Hierarchical Abstract Computers, *Comp. Lang.* 13, 109–123 (1988).
7. R. Adamov and L. Richter, A Proposal for Measuring the Structural Complexity of Programs, *J. Syst. Software* 55–70 (1990).
8. H. D. Rombach, A Controlled Experiment on the Impact of Software Structure on Maintainability, *IEEE Trans. Software Eng.* SE-13, 89–94 (1987).
9. H. D. Rombach, Design Measurement: Some Lessons Learned, *IEEE Software* 17–25 (1990).
10. S. Wake and S. Henry, A model based on software quality factors which predicts maintainability, in *Proceedings: Conference on Software Maintenance*, 1988, pp. 382–387.
11. S. R. Chidamber and C. F. Kemerer, Towards a metrics suite for object-oriented design, in *Proceedings: OOPSLA '91*, 1991, pp. 197–211.
12. T. McCabe and C. Butler, Design Complexity Measurement and Testing, *Commun. ACM* 1415–1424 (1989).
13. C. L. McClure, A model for program complexity analysis, in *Proceedings: 3rd International Conference on Software Engineering*, 1978, pp. 149–157.
14. S. N. Woodfield, Enhanced Effort Estimation by Extending Basic Programming Models to Include Modularity Factors, Ph.D. Thesis, Purdue University, West Lafayette, Indiana, 1980.
15. J. K. Kearney, R. L. Sedlmeyer, W. B. Thompson, M. A. Gray, and M. A. Adler, Software Complexity Measurement, *Commun. ACM* 1044–1050 (1986).
16. T. Korson and J. D. McGregor, Understanding Object-Oriented: A Unifying Paradigm, *Commun. ACM* 33, 41–60 (1990).
17. W. Li, Applying Software Maintenance Metrics in the Object-Oriented Software Development Life Cycle, Ph.D. Thesis, Virginia Polytechnic Institute and State University, Blacksburg, Virginia, 1992.