8-16-1996

Using Object-Oriented Design Complexity Metrics to Predict Maintenance Performance

Rajendra K. Bandi
Georgia State University, rbandi@gsu.edu

Using Object-Oriented Design Complexity Metrics to Predict Maintenance Performance

Rajendra K Bandi
College of Business Administration
Georgia State University
P.O. Box 4015, Atlanta, GA 30302-4015
e-mail: rbandi@gsu.edu

Introduction

The Object-Oriented (OO) paradigm has become increasingly popular in recent years. More and more organizations are introducing object-oriented methods and languages into their software development practices. Claimed advantages of OOP include easier maintenance through better data encapsulation (Booch, 1986). Some evidence has begun to appear that these benefits may be achieved in practice (Mancl & Havanas, 1990). Although maintenance may turn out to be easier for programs written in OO languages, it is unlikely that the maintenance burden will completely disappear (Wilde & Huitt, 1992). Maintenance, in its wildest sense of 'post deployment software support,' is likely to continue to represent a very large fraction of total system costs. Maintainability of software thus continues to remain a critical area even in the Object-Oriented era.

One approach to controlling software maintenance costs is the utilization of software metrics during the development phase. These metrics can be utilized as indicators of the system quality and can help identify potential problem areas. Several studies have been conducted examining the relationships between design complexity metrics and maintenance performance, and have concluded that design based complexity metrics can be used as predictors of the maintenance performance.

However, there are two general types of criticisms that can be applied to current software metrics. The first category is theoretical criticism. Traditional software complexity metrics do not possess appropriate mathematical properties, and consequently fail to display what might be termed normal predictable behavior (Weyuker, 1988).

The second category of criticism is more specific to OO design and development. The OO approach involves modeling the real world in terms of its objects, while more traditional approaches emphasize a function-oriented view that separates data and procedures. Because of the fundamentally different notions inherent in these two views, software metrics developed with traditional methods in mind do not lend themselves to notions such as classes, inheritance, encapsulation, and message passing (Chidamber & Kemerer, 1994). Therefore given that current software metrics are subject to criticism and are easily seen as not supporting key OO concepts, it seems appropriate to have new validated metrics especially designed to measure the unique aspects of the OO design.

The Research Problem

Information Systems researchers are showing a strong interest in the area of object-oriented metrics, as evidenced by the number of opinion papers arguing about the need for OO metrics. We have begun to see a few OO metrics emerging on the scene. However, very little work has been done in validating these metrics.

The research question that is the focus of this research is :

"Can we have complexity metrics designed to measure the unique aspects of the OO design, which not only meet the theoretical rigor for good metrics, but also are indicative of some quality of the software like maintainability"?

It is argued that the object-oriented paradigm encourages putting more effort into the design. More effort on design means more information with which design based metrics can be computed. However, most of the OO metrics proposed to-date do not exploit this additional information available at the design stage. We have come across research reported in two papers which take advantage of the additional information available in an OO design. Chen and Lu propose *Operation Argument Complexity (OAC)* and *Attribute Complexity (AC)* metrics which utilize the complexity of the arguments (parameters) of the class methods and the complexity of the class attributes (Chen & Lu, 1993). Abbott goes one step further and defines *Interface Size (IS)* and *Permitted Interaction Level (PIL)* metrics which are based on the notions of number and strength of interactions and interface parameters respectively (Abbott, 1993). The weakness of both of the above papers is that neither of them have empirically validated the metrics. The only validation done in both the cases is subjective validation, where the metrics values are compared to expert judgments. These four metrics OAC, AC, IS, and PIL are then the key metrics in our research.

3. The Design Complexity Metrics

Interface Size

The concept of interface size gives a measure of the means for information to flow in and out of their encapsulation. Some classes define many methods, perhaps many of which have complex signatures (i.e., parameter lists), provide abundant means for information to flow in and out of their encapsulation. Other classes may provide few methods, many of which have simple signatures. It is expected that larger interface size correlates with increased difficulty in comprehending how to select and correctly use the services provided by a class.

IS = {K1* (number of parameters) + K2* (sum of size of parameters)}

The constants K1 an K2 are tentatively set to 1 for simplicity.

Permitted Interaction Level

The concept of permitted interaction level specifies the amount of interaction that is permitted in a system, class, or a method. To explain further, whenever a method is invoked its parameters are used for some internal computation along with some of the data attributes associated with the class to which that method belongs. Also a value may be passed back to the caller. We say that there is an interaction between two entities A and B if the value of entity A is calculated based on the value of the entity B, or vice versa. In the context of the interaction level metric, if the value of some data attribute is calculated based on the value of one or more of the parameters, or vice versa, we say that there is an interaction between the parameters and the data attribute. It is expected that higher interaction level correlates with increased difficulty in determining how to implement or modify a design.

PIL = {K3* (number of interactions) + K4* (sum of strength of interactions)},

where strength of interaction is defined as the product of the "interface sizes" of the parameters involved in the interaction. The constants K3 an K4 are tentatively set to 1 for simplicity.

Operation Argument Complexity

The Operation Argument Complexity of a class is defined as (Chen & Lu, 1993): OAC = $_i$ P(i) where, P(i) is the size of each argument (parameter) in each operation (method) in the class. Summing up all P(i) in the class gives this metric value. The OAC metric would be equal to the Interface Size metric if the value of the constant K1 in the interface size metric is set to 0, and K2 is set to 1.

Attribute Complexity

The Attribute Complexity for a class is defined as (Chen & Lu, 1993): AC = $_i$ R(i) where, R(i) is the size of each attribute used in the class. Summing up all R(i) in the class gives the metric.

It must be noted that all the four metrics described above measure the within-class characteristics of a class. They do not measure any of the across-the-class characteristics like the relationships among the various classes. We realize that to fully understand the complexity of a system design we need to measure both *within-class* as well as *across-class* properties of the design. However in this study we are specifically focusing on the complexity due to *within-class* structure. Some metrics which would be useful in measuring the *across-class* properties include Coupling Between Classes, Number of Message Sends, etc.

The Research Approach

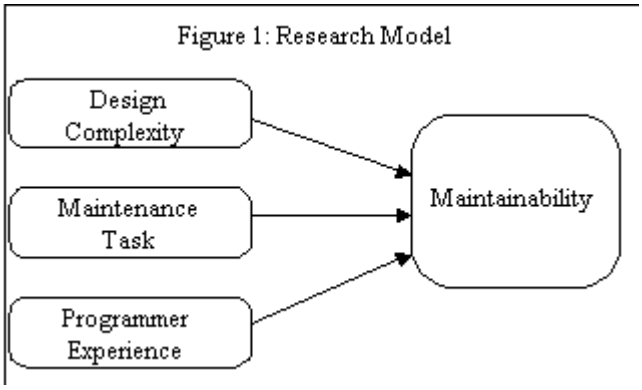
The metrics will be validated and refined using a dual approach. Both analytical and empirical validation techniques will be used to validate the metrics. Fenton proposes that a two-level metric validation scheme ensures that: (1) a metric is well-defined, consistent, and based on measurement theory; and (2) the metric is specifically related to or contributing to, a quality such as maintainability of the software (Fenton, 1990).

Research Model:

The model for this research is shown in figure 1. This research model is consistent with past research on maintenance performance.

Researchers generally incorporate one or more variables of the following types as independent variables (Gibson & Senn, 1989): Program measures, Programmer measures, Task measures. Each of these measures have been found to influence maintenance performance. Most researchers use system attributes like system complexity and/or size as program measures. In our research we use design complexity (measured by the four metrics) as the program measures. Programmer experience and programmer ability are considered important determinants of programmer performance, but experience is often used as a measure of programmer performance. In our research we use programmer experience as a programmer measure. It is a well known fact that task characteristics influence performance and hence is included in the model. Task characteristics are often considered as the type of maintenance task - perfective, corrective, adaptive etc. Maintainability is defined as the ease with which systems can be understood and modified, and is generally measured by the Time (required to implement changes) and Accuracy (of modifications) (Curtis et al., 1979; Gibson & Senn, 1989). Maintainability is measured in this research using the variables representing time and accuracy. Time to maintain is measured as the time taken per class to make changes. Accuracy is measured as the number of errors per class.

Figure 1: Research Model



Analytical Validation of the metrics:

This research proposes to use the list of properties suggested by Weyuker to analytically validate the metrics (Weyuker, 1988). This list, while currently still a subject of debate and refinement, is a widely known formal analytical approach. A similar approach is used by other OO metrics researchers (Chidamber & Kemerer, 1991).

Empirical Validation of the metrics:

This research proposes to empirically validate and refine these metrics and compare them with other well known OO metrics, using a controlled laboratory experiment, and follow it up with a field study. The metrics will be validated against maintenance performance (measured as time, and accuracy). This is consistent with the work done by previous researchers to examine the relationships between Design complexity, Code & Structure Complexities and Maintenance performance (Curtis et al., 1979; Gibson & Senn, 1989; Henry & Selig, 1990; Mancl & Havanas, 1990; Rombach, 1990; Li & Henry, 1993). The empirical validation will result in the following:

1. validated metrics
2. provide relationships between design complexity and maintenance performance
3. refinement of the metrics, by attempting to determine the values of some of the constants which have been tentatively set to a unit value.
4. comparing the metrics empirically with other well known non-detail-oriented OO metrics for predicting maintenance performance. These include metrics like Number of attributes, Number of methods, Depth of inheritance, Number of Children etc.

Importance of the Research

This research is motivated by the author's work in preparing OO metrics handbooks and makes important contributions to both the research and practitioner communities.

Contribution to IS Practitioners

Availability of validated design metrics provide senior managers (who may not be completely familiar with the design detail of an application) and designers with an indication of the quality of the design. The metrics can identify areas of the application that may require more rigorous testing and areas that are candidates for redesign. Thus potential flaws and other leverage points in the design can be identified and dealt with earlier in the life cycle.

Contribution to IS Researchers

This research makes important contributions to the IS research community. First, the research will result in theoretically evaluated set of complexity metrics for OO systems. Lack of theoretically sound metrics has been a major criticism of software metrics. Second, it will provide insights into the maintenance process of OO systems, and the role of design-based metrics to predict maintainability. This is an important contribution because while complexity metrics have been found to be able to predict the maintainability in traditional systems (Curtis et al., 1979; Gibson & Senn, 1989; Henry & Selig, 1990), we cannot simply extend the same argument to OO systems, because, it is known that OO systems have unique maintenance problems.

Current Status

A pilot experiment has been conducted and preliminary results are very encouraging. The complexity metrics show significant correlation with the time taken to maintain. The metrics however did not show any significant correlation with number of errors. Currently, the final details of the case to be used for the

laboratory experiment are being worked out. At the same time we are discussing with various organizations about collecting data for the field study.

References available upon request from author.