Managerial Use of Metrics for Object-Oriented Software: An Exploratory Analysis

Shyam R. Chidamber, David P. Darcy, and Chris F. Kemerer

Abstract—With the increasing use of object-oriented methods in new software development there is a growing need to both document and improve current practice in object-oriented design and development. In response to this need, a number of researchers have developed various metrics for object-oriented systems as proposed aids to the management of these systems. In this research an analysis of a set of metrics proposed by Chidamber and Kemerer [10] is performed in order to assess their usefulness for practicing managers. First, an informal introduction to the metrics is provided by way of an extended example of their managerial use. Second, exploratory analyses of empirical data relating the metrics to productivity, rework effort, and design effort on three commercial object-oriented systems are provided. The empirical results suggest that the metrics provide significant explanatory power for variations in these economic variables, over and above that provided by traditional measures, such as size in lines of code, and after controlling for the effects of individual developers.

Index Terms—Software metrics, object-orientation, productivity, effort, reuse, design, SLOC, WMC, NOC, DIT, LCOM, CBO, RFC, programmer, project management.

1 Introduction

There is great interest in use of the object-oriented approach to software engineering. This is due to a variety of claims about how it may improve the development of software, including such factors as greater reusability and increased extensibility. In order to help managers and developers achieve these goals, a variety of software metrics have been proposed to help track the status of software designs.

One such set of metrics is the set proposed by Chidamber and Kemerer (hereafter CK). A later paper refined the metrics and collected empirical metrics data at two sites, and presented the empirical distributions [10]. Interested readers should see the latter article for a detailed treatment of the metrics and their definitions.

Given the number of metrics that have been proposed, it seems appropriate that attention is being turned toward studying the degree to which various metrics actually meet their goal of providing insight into the design process. Sharble and Cohen report on how the CK metrics were used by Boeing Computer Services to evaluate different OO methodologies metrics [28]. Two implementations of an example system, one using responsibility-based methodology and another using data driven methodology were analyzed using these six metrics. Based on this analysis, Sharble and Cohen recommended the responsibility-based design methodology for use in the organization.

This suggests an active interest in the practitioner community to use well-constructed metrics as a basis for managerial decision-making. Li and Henry analyzed two commercial systems referred to as UIMS and QUES. They found that the CK metrics explained additional variance in maintenance effort beyond that explained by traditional size metrics [23]. This finding is significant in that some previous proposed complexity metrics in the procedural domain have been criticized as simply being surrogate measures for the size of the application [29].

Chidamber and Kemerer also report empirical data from two commercial organizations and suggest ways in which the metrics can be used to manage OO design efforts [10]. In that paper it was suggested that the primary use of the metrics by managers would be to identify outlying values that might reflect suboptimal design practice.

Most recently, Cartwright and Shepperd collected data from a telecommunications application in the U.K. [8]. Using a subset of the CK metrics, they found a positive correlation between the Depth of Inheritance Tree (DIT) metric and the number of user-reported problems. In Denmark, Soren Nielsen has also been exploring the implications of high CK metric values [24]. He has reported some preliminary results based on a study of a design model of 170 methods, 56 classes, and 150 attributes. Two classes were flagged as having very high CBO and LCOM values, and these classes were later identified as being the most difficult to successfully implement. Pant et al. collected data from a set of 69 public domain classes and 25 classes developed for their research on the generalization of OO components for reuse [25]. They found significant correlations of their tested metric with size measures such as SLOC and with previously developed complexity measures such as cyclomatic complexity [25, p. 26, Table 4].

Most recently Basili et al. have reported on results of using the CK metrics suite to predict the quality (fault proneness) of student C++ programs [2]. Five of the six metrics were shown to be useful predictors, and the suite of

- *S.R. Chidamber is with The Advisory Board Co., 600 Hampshire Ave., Washington, DC 20037.*
- *D.P. Darcy and C.F. Kemerer are with the Katz Graduate School of Business, University of Pittsburgh, Pittsburgh, PA 15260. E-mail: ddarcy@pop.pitt.edu, ckemerer@katz.business.pitt.edu.*

Manuscript received 4 Mar. 1997; revised 19 Dec. 1997.
Recommended for acceptance by D. Ross Jeffery.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number 104088.

design metrics was shown to be a more effective predictor of fault proneness than extant code metrics.

This paper is devoted to 1) providing further explanation of the CK metrics in a management, rather than a technical, context, and to 2) presenting exploratory data gathered from three OO financial application systems examining the relationships between these metrics and variables of managerial interest including productivity, re-use effort, and design effort. Section 1 will serve as a useful introduction to the metrics by way of an extended applied example of a type that was not presented in the first paper. Section 2 contains exploratory empirical results relating the metrics to variables of managerial interest. In particular, it is shown that the metrics provide useful information beyond that provided by size metrics alone. It should be remembered that such an approach is aimed at empirical validation rather than theoretical validation (for similar empirical work see [2]; for theoretical work, see [19], [7]).

The rest of the paper is organized as follows. Section 2 presents a simplified example of an OO design that is motivated by the systems that were analyzed as part of this study. This example will be used to illustrate the calculation of the CK metrics, and their potential for use during early design. Section 3 and Section 4 describe the actual empirical data collection, and present the exploratory results. A summary and concluding remarks are presented in Section 5.

2 A Simplified Trading System Example

2.1 A Base System (Version 1.0)

Consider a hypothetical OO system called TRADER that is built to assist a trader in a securities firm that buys and sells different financial instruments in the market place. A hypothetical system is used in this section only to illustrate use of the metrics, similar to [30]. In the later sections data from actual commercial systems in the finance domain are presented. This example is a highly simplified version of one of the actual systems (the "TPM" system, see "System Descriptions" section) that is analyzed later in this paper. The initial requirements for TRADER are to present the trader with a GUI screen and to gather the details of a transaction that is currently underway. Initially TRADER is designed to handle equity, bond and foreign exchange trading. TRADER will print individual transaction descriptions, and also consolidate in a database all the trades undertaken. In addition, TRADER has the ability to communicate with other systems to retrieve pertinent information (e.g., stock quotes, exchange rates). Fig. 1 shows the class hierarchy for key abstractions in the system. Fig. 2 shows the class structure, in pseudocode, for all four classes. A superclass trade is at the root, and there are three subclasses equity trade, foreign exchange (fx) trade and bond trade. The three subclasses are specialized cases of the superclass, and inherit all the methods and instance variables from it.

The six CK metrics will be illustrated below through this example. The metrics can be roughly calculated as follows: 1) the NOC metric is equal to the count of immediate subclasses, 2) WMC is the count of methods in a class (assuming unity weights for all methods), and 3) DIT is the length of the maximal path to the root of the class hierarchy [10]. The class *trade* has three children (*equity trade*, *fx trade*, and *bond trade*), three methods (*evaluate_counterparty*, *get_trade_id*, and *position_update*), and is a root class. Therefore, its NOC, WMC, and DIT metrics are 3, 3, and 0, respectively.

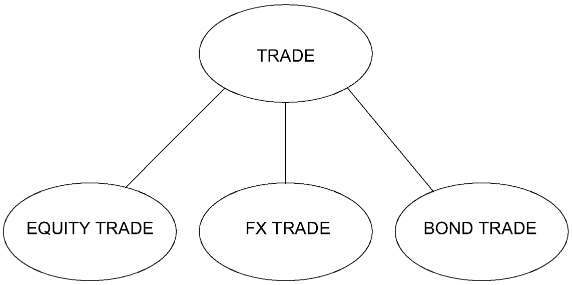


Fig. 1. Class hierarchy for the hypothetical TRADER system (Version 1.0).

CBO is approximately equal to the number of couples with other classes (where calling a method or instance variable from another class constitutes coupling), and RFC is the cardinality of the set of all methods that can execute in response to the arrival of a message to an object of a class. The CBO metric for *trade* is 1, since this class uses one method from another class (the *report_trade* method). The RFC metric is 4 since arrival of messages to this class could precipitate the possible execution of four methods (three declared in *trade* and the *report_trade* method called by *position_update*).

The LCOM metric is approximately equal to a count of the number of method pairs that do not have common instance variable minus the count of method pairs that do. Since there are three methods in *trade*, there are a total of three method pairs: *evaluate_counterparty & get_trade*, *evaluate_counterparty & position_update* and *get_trade & position_update*. The first and second pairs have no common instance variables. However, the third pair does have a common instance variable (*trade id*). Consequently, the number of disjoint method pairs is 2, the number of similar method pairs is 1, and therefore LCOM metric is 1 $\alpha = 2 - 1$). All six metrics for the four classes are shown in Table 1. Note that CBO is 2 for the class *equity trade*, since this class uses the *trade_id* instance variable and the *quotes* method from outside the class. Similarly, LCOM is shown as zero because it is assumed that *estimate_beta* and *get_stock_quotes* will both use a common instance variable.

TABLE 1
METRICS FOR TRADER CLASSES (VERSION 1.0)

Class Name	WMC	DIT	NOC	CBO	RFC	LCOM
trade	3	0	3	1	4	1
bond trade	1	1	0	0	1	0
fx trade	1	1	0	0	1	0
equity trade	2	1	0	2	3	0

2.2 An Enhanced System (Version 2.0)[1]

Now, consider the situation where bond trading operations are expected to include both municipal and corporate bonds. Municipal bonds have different interest rate and maturity calculations than corporate bonds, so the class hierarchy

[1] This is a simplified example drawn from the historical design evolution of the TPM system described later in the paper.

```
// class trade

Attributes // a.k.a. instance variables

trade id, counterparty, trade value // details of the trade

Operations // a.k.a. methods

evaluate_counterparty() // determines validity of the counterparty from a data
base file

get_trade_id() // obtain trade id details from the user

position_update ()

     call position_manager::report_trade()// send message to external
      position_manager class furnishing trade value and trade id.
----------------------------

// class bond trade

Attributes // a.k.a. instance variables

bond_details // bond ratings details

Operations // a.k.a. methods

get_bond_info() // access bond data base for current market price information
----------------------------

// class fx trade

Attributes // a.k.a. instance variables

forex_details // foreign market information

Operations // a.k.a. methods

calculate_exchange_rates() // access currency market monitor for latest rates
----------------------------

// class equity trade

Attributes // a.k.a. instance variables

company, stock_market, PE ratio, earnings, 52_week_hi&lo

Operations // a.k.a. methods

estimate_beta() // determine risk compared to rest of the market

get_stock_quotes() // consult external stock quotation data base

        call quotron::quotes() for trade_id // get the latest quote for the trade
```

Fig. 2. Pseudocode for classes in TRADER (Version 1.0).

could be changed so as to treat *municipal bond trade* and *corporate bond trade* as specializations of the class *bond trade*; each subclass can have separate calculation operations, but still share the attributes and operations that are declared in the class *bond trade.* Also, TRADER is expected to handle both domestic and international equities. Two new subclasses can be declared to specialize the class *equity trade.* The class declarations and new hierarchy are shown in Figs. 3 and 4. The metrics for the classes in the system are shown in Table 2.

Notice that CBO and RFC metrics for classes *corporate bond trade* and *international equity* are larger due to relying on obtaining the exchange rate information from the class *fx trade.* A careful designer would have realized that this information is no longer pertinent only to foreign exchange trading, and moved the rate calculation method to the class *trade* so that all types of trades could have easy access to it.[2] Instead, the short cut has resulted in coupling at the leaf level classes of the system. If the CK metrics were reported during a design review or system analysis report, the increase in CBO, RFC, and WMC metric values for leaf classes like *international equity* and *corporate bond* would indicate that perhaps encapsulation is being compromised, and opportunities for reuse via inheritance are being missed. Such compromises in design principles could lead to classes that are more difficult to understand, test, and reuse in another application. Notice the domestic equity class has mostly zeros in Table 2. As can be seen from the pseudocode, this is because this class remains largely unspecified. It is possible (so long as there were not too many instances of zeros) that a designer might find this information to be valuable in terms of identifying poorly specified classes, just as he or she would be expected to be interested in high metric values.

[2] For purposes of this example assume that the method is equally applicable to all of the classes.

```
// class municipal bond trade

// Attributes a.k.a. instance variables

state_or_federal, over_the_counter

// Operations a.k.a. methods

calculate_coupon_rate() // determine the inter-rate rate for the bond

     call Tbill_server::rates() // get the current rates from another source

----------------------------

// class corporate bond trade

// Attributes a.k.a. instance variables

adr, sp_rating

// Operations a.k.a. methods

calc_rating(sp_rating) // get the standard & poors rating

      if adr == TRUE then call fx trade::calculate_exchange_rates() // get the exchange rate information if this is a foreign bond issue

----------------------------

// class international equity

// Attributes a.k.a. instance variables

exchange_rate, quotation

// Operations a.k.a. methods

perform_analysis_roa (fx trade::calculate_exchange_rates()) // analyze the stock using the foreign exchange rate

get_quotron(quotation) // determine the current price of stock

----------------------------

// class domestic equity

// Attributes a.k.a. instance variables

attribute1

// Operations a.k.a. methods

none defined yet
```

Fig. 3. Pseudocode for classes in TRADER (Version 2.0).

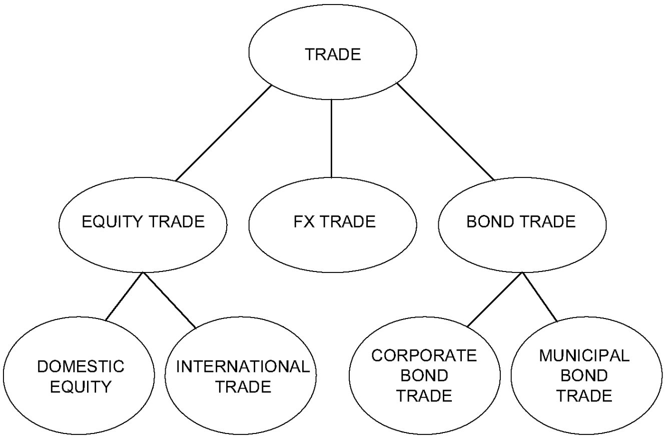


Fig. 4. Class hierarchy for the hypothetical TRADER system (Version 2).

TABLE 2
METRICS FOR TRADER CLASSES (VERSION 2.0)

Class Name	WMC	DIT	NOC	CBO	RFC	LCOM
trade	3	0	3	1	4	1
bond trade	1	1	2	0	1	0
fx trade	1	1	0	0	1	0
equity trade	2	1	2	2	3	0
municipal bond trade	1	2	0	1	2	0
corporate bond trade	1	2	0	1	2	0
international equity	2	2	0	1	3	1
domestic equity	0	2	0	0	0	0

Summary. This simple example demonstrates: 1) how the metrics are calculated for individual classes, 2) that the metrics change as the design progresses over time, 3) that even a rudimentary system can quickly develop unnecessary complexity that detracts from the architectural integrity of the design, and 4) that metrics can be used during the design process to reduce developer controllable complexity.

3 Exploratory Validation Study

3.1 Introduction

The reduction of developer controllable complexity is of considerable importance to a software project manager, since researchers have demonstrated its crucial impact on the typical managerial concerns of cost, quality, and productivity [1]. Since the CK metrics were developed to be measurements of design complexity, variations in productivity, rework effort, and design effort may be expected to be explained, in part, as a function of high metric values [18], [28], [10], [2], [8]. Data were collected from three commercial systems in order to examine the relationships between the CK metrics and the above variables. Such an analysis is designed to provide evidence on the degree to which each complexity metric influences a dependent variable. Results of such an analysis can be used by managers to make more informed design and resource allocation decisions in organizations, such as estimating the development and maintenance costs of a piece of software.

3.2 Data Collection Requirements

The primary criterion for a data site for this research was that the organization had to be actively undertaking object-oriented development for production systems. Also, the data collection needed for metrics validation requires active collaboration from sites with at least rudimentary data collection in place. In addition to these data requirements, the site needed to be willing to make design documents, project plans and source code available for inspection so that multiple sources of data are available to measure the model variables. Given all of these constraints, it was decided to study several projects over a period of time at a single data site rather than attempt a cross-sectional study of many organizations. Past research that has validated metrics on commercial software systems has often adopted a similar strategy[15], [1], [23].

It cannot be overemphasized that, given the nascent state of OO metrics research, the analysis presented in this paper is exploratory in nature, and it would be unwise to overgeneralize from the empirical estimates presented below. Nevertheless, these analyses are presented as real world examples of how the metrics may be applied and the types of insights that metrics can provide.

3.3 Background of Data Site

The Information Systems Division (ISD) of a European bank (referred to in this paper as European Bank Corporation, or EBC) was selected as the data site for this research. EBC is an international bank that offers a wide range of modern financial instruments to clients on a worldwide basis. Technological leadership is central to their competitive position in the financial services industry. The ISD staff of over 130 people is charged with the responsibility of providing computer hardware and software solutions that provide access to up-to-the-minute information, analytical capabilities and financial engineering tools to traders and portfolio managers in the bank. These solutions help to formulate portfolio structure, assess market risk factors, advise market makers, and ensure liquidity of financial positions and competitive pricing of financial instruments. The strategic and financial implications of the transactions that are handled by these systems are significant, and therefore the software systems are planned, designed and tested with considerable care. The organization has had a tradition of collecting detailed data during various phases of the systems development process to facilitate management control of IS projects.

3.4 System Descriptions

Three software systems developed by the ISD at EBC were analyzed as part of this study. (Their names are disguised and only basic information about their functions is provided in order to preserve the data site's anonymity.) These systems are used by financial traders to assist them in the buying, selling, recording, and analysis of various financial instruments like stocks, bonds, options, derivatives, and foreign exchange positions. All three projects were completed in the 1990s and are used extensively by EBC traders in the United States, Europe, and Asia. (See Table 3.)

TPM (Trade Position Manager). TPM is a key software system in the IS infrastructure at EBC. It is responsible for "straight through" trading, i.e., it enables computers from the trading desk to communicate to back-office mainframe computers that maintain and consolidate the massive amount of data relating to the bank's current and historical trading in financial instruments. TPM manages the physical and logical distribution of data among different computer systems, while providing users with a single consistent database for product risk assessment and position keeping. TPM was developed on a Sun SparcStation in C++ using the Booch OO design methodology [6], and is comprised of several key subsystems that were designed by a team of six designers over a period of 8 months. This is the system that the simple example at the beginning of the paper is based upon.

FIS (Fixed Income Sales). FIS is a trading decision support system. It provides pricing, analytics,[3] risk management and management information to the Fixed Income Trading and Sales teams in the investment bank. In addition to these functions, FIS automates the manual process of issuing "trading tickets." Traditionally, each time a trader completes a trade, the details of the asset (stock, bond) and the transaction are recorded on a slip of paper known as a "ticket." FIS also passes the consolidated information to other mainframe computer programs for initiating electronic transfer of funds, archival and storage. FIS involved five designers, and there are a number of different versions of the system that have been developed over the period 1991-1993. It was developed using Objective C and works primarily on NeXT workstations. Certain classes developed for FIS were considered to be reusable in other systems, and the data available for collection during this project were restricted to the classes that were reused in another project.

SLB (Securities Lending and Borrowing). The business mission of SLB is to help maximize the returns on trading positions by effectively coordinating the borrowing and lending of equity and equity-linked instruments. The system assists securities traders in an investment bank to lend to and borrow

[3] This is a 'term of art' within the finance industry. It means providing shell analytical tools without the data. The user supplies the data.

TABLE 3
SUMMARY OF APPLICATION SYSTEMS

System	Metrics Data Source	No. of classes	Dependent Variable
TPM	15,096 LOC of C++ code	45	productivity
FIS	2,705 LOC of Objective C code	27	rework effort
SLB	Design documents	25	design effort

from external counterparties (other financial institutions) to finance their trading positions. The SLB system provides functionality to enter stock borrowings and loans into a database, checking market price of stocks, tracking pending loans and payments, and generating accounting and financial reports. The system provides a graphical user interface and analysis tools and the capability to communicate with other computer systems to access and consolidate data. Two people were assigned to work full-time on the design of classes for this system. They produced about 40 pages of design documents in about 3 weeks.

One aspect of the SLB system worth noting is that it provided a confirmation for a claim made in earlier work [10]. In that article it was argued that the six metrics were design metrics and could be collected from design documents. However, this argument was not *directly* supported in [10] as the two systems represented in that paper had both been implemented, and therefore the program code was used for the analysis. Here the metrics data from the SLB system were extracted from systems analysis and detailed design reports that were issued *prior* to coding. Therefore, it is clear that these data reflect design phase data, although of a detailed nature.

3.5 Data Sources

Source Code Control Systems (SCCS). SCCS is a source management system used at ISD that maintains records of changes made in files within a project. Records stating what the changes were, why and when they were made, and who made them are kept for each version. Previous versions can be recovered, and different versions can be maintained simultaneously. Since all systems at EBC employ the SCCS utility, detailed records of time, version, size, and programmer usage were made available for this research.

Project Management Documents. EBC has formal mechanisms for project initiation, approval, and management. There are steering committees that oversee individual projects and several documents are released during different stages of a software products' life cycle. Two such documents are the requirements document and the systems analysis and design report. Both these documents contain information on the cost (in terms of time), and early design descriptions of key classes that are used in the system. In addition the project manager in charge of each project releases detailed project plans that describe current design status, staffing levels and costs. These documents were made available for the research study to aid data collection.

Programmer Reports. In addition to the formal records maintained in SCCS and in the project management documents, individual programmer activity reports were another source of data for this study. This helped in discounting personal holidays and time taken to attend to activities that are not directly related to the project. In the case of all three projects key designers also reviewed their personal calendars to further ensure data accuracy.

3.6 Data definitions

In addition to the six metrics, WMC, DIT, NOC, RFC, CBO, and LCOM, the following variables were used in the analysis of the three systems in this study:

SIZE. Size of class in lines of code defined in that class (i.e., does not include LOC in any associated super or subclasses). This definition excludes header files but includes comments.

EFFORT. Amount of developer time, measured in hours.

PRODUCTIVITY. Productivity measured as SIZE/EFFORT [20].

STAFF_*n*. Dummy variables identifying specific developers used as control variables in the models.

4 EXPLORATORY RESULTS

Section 3 described the variables that were measured from the three systems at EBC. Productivity, rework effort, and design effort are the key dependent variables that are examined, as appropriate for individual systems. In Section 4, three categories of regression models using these variables are presented, along with an interpretation of the various models.

The primary analysis technique used in the analysis is stepwise regression. This is an exploratory technique whereby all the independent variables are allowed to enter and leave the model. After several iterations of adding or removing one variable, a model is presented to which adding any more variables will not significantly increase the degree of variation that the model explains (i.e., R^2) and removing a variable will significantly reduce R^2. Interested readers are referred to [26] or other standard text for further details.

4.1 General Results (Three Data Sets)

One of the first empirical results are the generally small values for the NOC and DIT metrics in all three systems, thereby reducing their chances of explaining variance in the dependent variables (see Tables 4, 5, 6, and 7). It is interesting to note that two earlier (and significantly larger) systems examined at different data sites also had relatively little use of inheritance as reflected by their NOC and DIT metric values [10]. And, most recently, Cartwright and Shepperd report almost no use of inheritance, as measured by DIT and NOC, at their U.K. telecommunications site [8].

This apparent lack of use of inheritance could be a result of two possible events. One explanation would be that this is an accidental failing to use a feature which is deemed to be desirable. If this is the case, then for OO designers and methodologists this may represent a growing set of documented cases of how OO adopters are not using the inheritance feature of OO. For managers of OO projects and for senior management, it illustrates how care must be taken to actively manage projects in such a way as to achieve promised benefits. Part of this is the illustration of how difficult it is to achieve certain results (such as a stated level of inheritance) without a measurement system that records those values. And, of course, without a means of determining the values, incentive systems to encourage the results are impossible. Alternatively, the lack of inheritance could reflect a conscious decision not to use a feature for which some believe that costs, in terms of complexity and maintainability, are deemed to outweigh the benefits. This was the case at the current data site [9]. This had also been reported at an earlier data site [10, p. 486].

However, here it is simply noted that, regardless of opinion on this issue, a manager should be interested in measuring the degree to which it is present in their applications. From the perspective of the current analysis, DIT and NOC have been omitted from possible inclusion in the models, given the small numbers of classes that took advantage of this feature.

A second general result that is able to be commonly analyzed across the three data sets is that the remaining metrics exhibit some potential problems with multicollinearity since WMC, RFC, CBO are highly correlated (most greater than 0.8).[4.] This suggests that, although WMC, RFC, and CBO measure different aspects of class design, there is a significant statistical reason to believe that classes with high (or low) WMC also have high (or low) values of RFC and CBO in this particular organization. Consequently, simultaneous use of these three variables in a regression analysis would generate coefficient estimates that are difficult to interpret. This underscores the possibility that only subsets of the six metrics may explain variance in the dependent variables at different data sites. In particular, it might be expected that only one metric of the set WMC, RFC, CBO would appear in any model. The correlations between RFC, WMC, and CBO for the three systems can be seen in Table 4.

Third and finally, recall the original premise behind managerial use of the metrics as suggested in [10]. The premise was that extreme (outlying) values signal the presence of high complexity that may require management action, e.g., assigning a higher skilled developer to that implementation, assigning extra testing resources to that component, etc. This approach has been generally accepted in the software metrics literature, for example, see the early work of Harrison [16], and recently, the OO metrics work of Henderson-Sellers [17]. This is not only intuitive, it is also pragmatic, as a measurement tool that suggested 'looking at everything' would have insufficient discriminant ability to be of value. Therefore, the relationship focused on in the main analyses is that between a management oriented dependent variable and outlying values of the complexity metrics. The threshold values of the metrics could not be determined a priori, given the novelty of these data, but rather will come from each of the three data sets, which is consistent with the exploratory approach of this research. However, a lower bound was defined for all data sets at 20 percent; i.e., the cutoff value for defining a "high" value for a metric can be defined no lower than the 80th percentile. While no hard and fast rules exist for such cutoffs, it was believed that this was consistent with the common Pareto (80/20) heuristic (see, for example [27]).

4.2 TPM Exploratory Analysis Results

The summary statistics for this data set are presented in Table 5.

Since this system was developed "from scratch" rather than a result of reuse, the dependent variable of interest for this system is productivity, defined as size divided by the number of hours required (effort). As a control feature, an examination of the first order Pearson correlations among the variables suggests that productivity is significantly (0.05) positively correlated to SIZE and STAFF_4 (dummy variable for a specific programmer). This suggests that these two variables are candidates for a base model for analysis of the marginal explanatory power of the six metrics, i.e., the extent to which the metrics explain variance in the dependent variable over and beyond traditional size measures and taking into account individual developer ability differences [3], [15]. This approach is similar to the one adopted by Banker et al. in demonstrating the explanatory power of complexity metrics in a traditional development environment, and of Li and Henry in an OO environment [1], [23]. Note that it also takes into account the suggested effect of variances in individual performance, a factor that is much discussed in practice, but relatively rarely accounted for in the research literature [12], [21], [22], [14].

The results of the base model are:

$$\begin{aligned}\text{PRODUCTIVITY} &= 1.78 + 0.06 * \text{SIZE} + 64 * \text{STAFF\_4}\\ &\quad (t = 0.44)\ (t = 6.16)\ (t = 6.92)\end{aligned}$$

The t statistics demonstrate the significance of the estimate of the coefficient. Except for the intercept value of 1.78 (t = 0.44), all of the above t statistics show that the coefficients are significant at the 95 percent level ($\alpha = 0.05$) [26].

The adjusted R-squared for this model was 0.75, and both predictors are significant at the 0.001 level (n = 45). The model passes the Belsley-Kuh-Welsh (BKW) test for multicolinearity among the independent variables [4].[5] The interpretation is that productivity, as defined here, was positively associated with size in LOC and with developer no. 4, whose work, all else being equal, was more productive than others on this project. These two variables are retained in the full model to reduce the possibility that significant results will be attributed to the complexity metrics due to the effect of what would otherwise (without the controls) be omitted variables.

Dummy variables were created out of each of the complexity variables, with labels such as "HICBO," "HILCOM," etc. These variables take on the value of one or zero, depending on whether the metric value exceeds the threshold value for the data set. The threshold values were set so as to accommodate

[4] Recall that in [10, p. 487] it was casually observed that some classes with high values of one metric often seemed to have high values of other metrics as well. An interpretation of this was that there may be a small number of highly complex or ill-designed classes in an application. The current correlation analysis may be further evidence for this phenomenon.

[5] The BKW test was run for all regressions reported in this paper, and this result holds in all cases.

TABLE 4
CORRELATIONS FOR RFC, WMC, AND CBO

	TPM			FIS			SLB		
	RFC	WMC	CBO	RFC	WMC	CBO	RFC	WMC	CBO
RFC	1 (0.0000)	–	–	1 (0.0000)	–	–	1 (0.0000)	–	–
WMC	0.98 (0.0001)	1 (0.0000)	–	0.28 (0.1509)	1 (0.0000)	–	0.95 (0.0001)	1 (0.0000)	–
CBO	0.95 (0.0001)	0.88 (0.0001)	1 (0.0000)	0.80 (0.0001)	–0.11 (0.5957)	1 (0.0000)	0.87 (0.0001)	0.95 (0.0001)	1 (0.0000)

(cells show Pearson correlations, p-values are in parenthesis).

TABLE 5
SUMMARY STATISTICS FOR TPM DESIGN AND MAINTENANCE (n = 45 CLASSES)

Variable	Median	Mode	Mean	Max	Standard Deviation
WMC	6	0	9.27	63	11.60
DIT	0	0	0.04	2	0.30
NOC	0	0	0.07	2	0.33
RFC	7	0	13.82	102	18.27
CBO	2	0	4.51	39	7.21
LCOM	0	0	6.96	90	16.46
PRODUCTIVITY	19.7	11.75	28.36	198	32.91
SIZE	267	66	335.47	1,308	263.69

natural breaks in the data at or above the 80/20 value, and vary across the three data sets.

For example, for TPM,

$$\text{HILCOM} = 1 \text{ if LCOM} >= 40, \text{ and } 0 \text{ otherwise.}$$
$$\text{HICBO} = 1 \text{ if CBO} >= 25, \text{ and } 0 \text{ otherwise.}$$

In the discussion section managers are cautioned against adopting these site-specific values without local calibration.

Using stepwise regression, the metric variables are allowed to enter the model. The analysis yields the following result:

$$\begin{aligned}\text{PRODUCTIVITY} = {}&–6.41 + 0.10 * \text{SIZE} + 48.11 * \text{STAFF\_4}\\ &(t = –1.65)\quad (t = 8.27) \qquad (t = 5.61)\\ &– 76.57 * \text{HICBO} – 33.96 * \text{HILCOM}\\ &(t = –4.03) \qquad (t = –3.00)\end{aligned}$$

Except for the intercept all of the above t statistics show that the coefficients are significant at $\alpha = 0.05$. The adjusted R-squared = 0.82, suggesting a good fit with the data. The model supports the notion that developers tended to be less productive, i.e., require more hours per line of code, for classes with high values of either CBO or LCOM. Dummy variables for WMC and RFC were not statistically significant at usual levels.

The intuition behind the significance of CBO in this model is that since classes that have a large number of interconnections with other classes are likely to require greater understanding of architectural details like the inheritance hierarchy and message passing, it can be hypothesized that they will take more time to develop, test and modify [6]. Consequently the productivity levels for such classes may be lower. The CBO metric is a measure of interclass coupling, and therefore may reflect this.

The traditional notions of cohesion measure the interrelatedness between portions of a program. The LCOM value provides a measure of the relative disparate nature of methods in the class. A smaller number of disjoint pairs implies greater similarity of methods and greater cohesion. Conversely, a high LCOM value indicates disparateness in the functionality provided by the class. Such a class may be attempting to achieve many different objectives, and consequently may behave in less predictable ways than classes that have lower LCOM values. Such classes could be more error prone and more difficult to test, and the data suggest that they require more effort to develop. It might be expected that high values might be more likely early in the design process, and signal that the class design is not yet well-understood.[6]

The results for CBO and LCOM are consistent with results in software engineering that higher coupling and lower cohesion are detrimental to productive development (see, for example [5]). In addition, they do so in a model where the portion of the variance in productivity that might be explained by simple size measures and by the inclusion of a variable reflecting individual ability variances has already been allocated. The two complexity metrics explain additional variance over and beyond these two commonly accepted factors.

An analysis of the residuals was conducted using both Cook's and White's tests [11], [31], respectively. No significant heteroskedasticity was detected. In addition, in examining the data it was noted that there was a single significantly outlying value for the dependent variable, productivity. The model was rerun without this observation and no significant changes were observed, enhancing confidence in the results.

A further sensitivity analysis was performed, allowing HIDIT and HINOC to enter the model. Neither did, leaving the results invariant to the original specification.

[6] We are indebted to Soren Nielsen for this latter insight.

4.3 FIS Exploratory Analysis Results

Since one of the major benefits of the OO paradigm is the ability to reuse key classes across applications, reusability is a key variable of managerial interest. Reusable classes can be a key to faster development cycles. In theory, a designer could assemble pre-existing classes, and build a new system in a fraction of the time it would take to build it anew. However, it is rare that a class created in one application would be directly reusable in another with zero rework. This rework effort was recorded for the classes from the FIS project that were reused on another project. The rework effort was measured in the number of hours spent by the next project staff to modify each class for use in the new project. The summary statistics for these variables is presented in Table 6.

TABLE 6
SUMMARY STATISTICS FOR THE FIS DATA SET (n = 27 CLASSES)

Variable	Median	Mode	Mean	Max	Standard Deviation
WMC	22	22	20.22	31	7.34
DIT	1	1	1.11	2	0.32
NOC	0	0	0.07	2	0.38
RFC	33	22	38.44	93	23.74
CBO	7	6	8.63	22	4.45
LCOM	0	0	29.37	387	90.72
SIZE	101	94	100.19	153	25.42
EFFORT	1	0.15	4.41	16	6.05

Similar to the process described above, a stepwise regression was performed and yielded the following model:

$$\text{REWORK\_EFFORT} = \underset{(t = 2.54)}{2.09} + \underset{(t = -2.34)}{-10.48} * \text{STAFF\_1} + \underset{(t = 3.07)}{7.43} * \text{HICBO} + \underset{(t = 3.28)}{8.95} * \text{HILCOM}$$

The adjusted R-squared for this model was 0.60, and the first two estimated coefficients are significant at the $\alpha = 0.05$ level, and with the metric variables being significant at the $\alpha = 0.01$ level. Just as with the TPM productivity data, high levels of both CBO and LCOM were associated with managerially poorer results, in this case higher rework effort.

An analysis of the residuals was conducted using both Cook's and White's tests, and no significant heteroskedasticity was detected.

As with the TPM analysis, these results were statistically significant in explaining variance over and above that already explained by conventionally accepted variables such as individual programmer differences or the number of lines of code for the class. However, unlike the TPM model, the FIS model consists entirely of dummy variables, limiting the total number of unique combinations. Since this is relatively less attractive if there was a desire to actually interpret the values of the estimated coefficients, a sensitivity analysis was conducted where the size variable was added to the above model. The following result is obtained:

$$\text{REWORK\_EFFORT} = \underset{(t = 1.09)}{3.75} + \underset{(t = -1.97)}{-9.61} * \text{STAFF\_1} + \underset{(t = 2.82)}{7.13} * \text{HICBO} + \underset{(t = 3.26)}{9.26} * \text{HILCOM} + \underset{(t = -0.50)}{-0.01} * \text{SIZE}$$

Although the metric value coefficients are still significant at an $\alpha = 0.01$ level, the coder variable is no longer significant at an $\alpha = 0.05$ level and the size variable is similarly insignificant. Understandably, the adjusted R-squared for the model drops.

An additional sensitivity analysis was run, allowing HIDIT and HINOC to enter the model. In this case, somewhat surprisingly, HINOC does replace HICBO in the model. Upon close examination, this is a result of the idea, reported above, that the metric values tend to be correlated, i.e., a small number of classes tend to have high values of more than one metric (the "80/20" rule). Specifically in the case of FIS, HILCOM, and HICBO share many observations in common, as do HICBO and HINOC. Therefore, with HILCOM in the model, HICBO and HINOC become close substitutes for this data set. This variance is no doubt a consequence of the exploratory nature of the analysis and the relatively small data sets.

4.4 SLB Exploratory Analysis Results

The models tested thus far have utilized data gathered from source code from the TPM and FIS projects. The SLB project data set is significantly different from the preceding two in that the metrics data have been gathered prior to writing any code. The metrics were obtained from the output of a Joint Applications Development (JAD)-like session where major application classes are specified, but not coded [13]. Besides the metrics data, the amount of time spent (in hours) to specify the high level design of each of the classes was also collected, although individual staff names were not available. Table 7 provides the summary statistics.

TABLE 7
SUMMARY STATISTICS FOR SLB (n = 25 CLASSES)

Variable	Median	Mode	Mean	Max	Standard Deviation
WMC	5	5	6.48	22	3.93
DIT	2	2	1.96	3	0.73
NOC	0	0	0.92	11	2.43
RFC	7	5	9.80	42	8.67
CBO	0	0	1.28	8	2.21
LCOM	0	0	4.08	83	16.65
EFFORT	4	4	4.20	24	6.41

As this analysis was performed on an early design, there are no lines of code data to use as a SIZE variable for a base model. Therefore, there is only a final model, developed in a similar fashion to the TPM and FIS models:

$$\text{DESIGN EFFORT} = \underset{(t = 2.25)}{2.05} + \underset{(t = 3.56)}{7.95} * \text{HICBO} + \underset{(t = 3.07)}{14.00} * \text{HILCOM}$$

The adjusted R-squared for this model was 0.60, and both of the estimated coefficients for the metrics are significant at the $\alpha = 0.01$ level. One interesting aspect of this model is that it explains about as much variance in design effort for SLB as the model of rework effort did for FIS, but without the two extra variables of size and staffing. Again, as in the FIS model, the number of unique combinations is quite small, which should serve to discourage direct interpretation of the estimated coefficients.

An analysis of the residuals was conducted using both Cook's and White's tests, and no significant heteroskedasticity was detected.

A further sensitivity analysis was performed, allowing HIDIT and HINOC to enter the model. Neither did, leaving the results invariant to the original specification. Therefore, of the six possible opportunities for DIT and NOC to be included in the models (two variables times three systems), in only one case, NOC for FIS, were they included.

5 Summary and Discussion of Results

The empirical results across the three financial services application can be summarized as follows:

- Useful metrics data could be collected on systems that were written in a variety of programming languages and on a system that was not yet coded;
- None of the three applications at this site showed significant use of inheritance—DIT and NOC tended to have minimal values;
- WMC, CBO, and RFC tended to be highly correlated;
- High levels of coupling (HICBO) and lack of cohesion (HILCOM) were associated with:
  - lower productivity
  - greater rework
  - greater design effort.

These results should be of interest to both practitioners and the OO research community as to the degree to which the metrics are of practical significance.

The low values of DIT and NOC indicate that reuse opportunities (via inheritance) are perhaps being compromised in favor of comprehensibility of the overall architecture of the application. In such cases, the low metric values may indicate that appropriate design preferences are being followed. It is important to note that, although generally not able to be demonstrated in the regression models due to their low variability, these two metrics can, in theory, be helpful in highlighting egregious departures from design principles (either higher or lower usage of inheritance). This is particularly relevant in projects that are staffed with programmers who are less familiar with object-oriented design.

The high degree of correlation among WMC, CBO, and RFC may explain why only CBO was a statistically significant predictor in these models. Once the CBO dummy enters the model, there may be little 'space' for WMC and RFC. This may suggest that a smaller set of hybrid metrics would have the same degree of predictive power. However, at this early stage in the development and adoption of Object-Oriented Design, having alternative metrics may compensate for the multicollinearity problem by being more meaningful to designers and managers in their work. Also, the correlations found here need to be replicated elsewhere before their degree of independence can be determined with confidence.

In each of the three systems, the stepwise regression analysis showed CBO and LCOM as being significant explanatory variables. It is important to emphasize the strength of this result given that each of these models had a different dependent variable, i.e., productivity for TPM, rework effort for FIS and design effort for SLB. It is interesting to note that these results are consistent with preliminary results found elsewhere, and may reflect the strength of the underlying concepts of coupling and cohesion [24].

As suggested in [10], [17], the metrics can be used to flag outlying classes for special attention. Exactly what constitutes an outlying class is an empirical question, but clearly is in part a function of the degree of certainty or risk for which the manager wishes to control. For example, in some applications, a strong requirement for low testing and maintenance costs would argue for paying extra managerial attention to a quite significant fraction of the classes. Alternatively, a very large application or one with less stringent requirements might dictate that only a smaller fraction of classes could or should be given special attention. So, the first question to be answered by the manager is what percentage of the classes merit examination. This may dictate the appropriate threshold values to be used, whether they are 20 percent or something else.

Going further, at this early stage in OO metrics research it is not known whether there may be characteristics of environments or application types that will tend to generate higher or lower values for the metrics. Therefore, practicing managers are urged not to accept the values reported in the TPM analysis section as rules, but rather to analyze local data and to set thresholds appropriately as described above.[7]

Of course, a certain degree of caution needs to be exercised overall in the interpretation and use of the metrics as levers of managerial control. The data sets have small sample sizes, which reduces the statistical power and reliability of the estimates. Additionally, all the data sets are drawn from a single organization that develops software systems for the financial services industry, which may limit the applicability of the specific statistical estimates to other general OO environments.

The CK set of metrics has been shown by this empirical validation to offer quantitative and significant insight into the impact of OO design decisions on managerially relevant variables of productivity and cost. It is hoped that this research helps in moving software development management in general, and OO software in particular, toward a more scientific basis.

Acknowledgments

The authors gratefully acknowledge helpful comments from participants at the Bournemouth, United Kingdom Metrics workshop and at the University of Pittsburgh MIS seminar. Helpful comments were also received from Michelle Cartwright, Rachel Harrison, Brian Henderson-Sellers, D. Ross Jeffery, Laurie Kirsch, Colin Kirsopp, Helen Klein, Mayuram Krishnan, Soren Skogstad Nielsen, Thomas Potok, Martin Shepperd, David P. Tegarden, and two anonymous referees.

[7] For example, it would be considered extremely unfortunate if these values were reported on in the manner in which some readers reported on the cyclomatic complexity limit of 10 when that metric was first introduced.

REFERENCES

[1] R.D. Banker, S.M. Datar, C.F. Kemerer, and D. Zweig, "Software Complexity and Software Maintenance Costs," *Comm. ACM*, vol. 36, pp. 81-94, 1993.

[2] V.R. Basili, L.C. Briand, and W.L. Melo, "A Validation of Object-Oriented Design Metrics as Quality Indicators," *IEEE Trans. Software Eng.*, vol. 22, pp. 751-761, 1996.

[3] V.R. Basili and R.W. Selby, "Calculation and Use of an Environment's Characteristic Software Metric Set," *Proc. Eighth Int'l Conf. Software Eng.*, pp. 386-391, 1985.

[4] D.A. Belsley, E. Kuh, and R.E. Welsch, *Regression Diagnostics*. New York: John Wiley & Sons, 1980.

[5] E.V. Berard, *Essays on Object-Oriented Software Engineering*. Englewood Cliffs, N.J.: Prentice Hall, 1993.

[6] G. Booch, *Object Oriented Analysis and Design*. Redwood City, Calif.: Benjamin / Cummings, 1993.

[7] L.C. Briand, S. Morasca, and V. Basili, "Property-Based Software Engineering Measurement," *IEEE Trans. Software Eng.*, vol. 22, pp. 68-86, 1996.

[8] M. Cartwright and M. Shepperd, "An Empirical Investigation of Object-Oriented Software in Industry," Technical Report TR 96/01, Dept. of Computing, Talbot Campus, Bournemouth Univ. 1996.

[9] S.R. Chidamber, "Metrics for Object-Oriented Software Design," PhD thesis, MIT, 1994.

[10] S.R. Chidamber and C.F. Kemerer, "A Metrics Suite for Object-Oriented Design," *IEEE Trans. Software Eng.*, vol. 20, pp. 476-493, 1994.

[11] R.D. Cook, "Influential Observations in Linear Regression," *J. Am. Statistical Assoc.*, vol. 74, pp. 169-174, 1979.

[12] B. Curtis, "Substantiating Programmer Variability," *Proc. IEEE*, vol. 69, p. 846, 1981.

[13] E.J. Davidson, "An Exploratory Study of Joint Application Design (JAD) in Information Systems Delivery," *Proc. 14th Int'l Conf. Information Systems*, pp. 271-283, Orlando, Fla., 1993.

[14] A.M. Davis, "Eras of Software Technology Transfer," *IEEE Software*, vol. 13, pp. 4-7, 1996.

[15] G.K. Gill and C.F. Kemerer, "Cyclomatic Complexity Density and Software Maintenance Productivity," *IEEE Trans. Software Eng.*, vol. 17, pp. 1,284-1,288, 1991.

[16] W. Harrison, "Using Metrics to Allocate Testing Resources in a Resource Constrained Environment," Technical Report 90-5, Dept. of Computer Science, Portland State Univ. Apr. 1990.

[17] B. Henderson-Sellers, "Object-Oriented Metrics: Measures of Complexity," Upper Saddle River, N.J.: Prentice Hall, 1996.

[18] S. Henry and W. Li, "Metrics for Object-Oriented Systems" *Proc. OOPSLA'92 Workshop: Metrics for Object-Oriented Software Development*, Vancouver, Canada, 1992.

[19] M. Hitz and B. Montazeri, "Chidamber and Kemerer's Metrics Suite: A Measurement Theory Perspective," *IEEE Trans. Software Eng.*, vol. 22, pp. 267-271, 1996.

[20] D.R. Jeffery and M.J. Lawrence, "Managing Programming Productivity," *J. Systems and Software*, vol. 5, pp. 49-58, 1985.

[21] C.F. Kemerer, "Bridging the Gap Between Research and Practice in Software Engineering Management: Reflections on the Staffing Factors Paradox," H.D. Rombach et al., eds., *Experimental Software Engineering Issues: Critical Assessment and Future Directions*. Berlin: Springer-Verlag, 1993.

[22] C.F. Kemerer and M.W. Patrick, "Staffing Factors in Software Cost Estimation Models," J. Keyes, ed., *Software Engineering Productivity Handbook*. New York: McGraw-Hill, 1993.

[23] W. Li and S. Henry, "Object-Oriented Metrics that Predict Maintainability," *J. Systems and Software*, vol. 23, pp. 111-122, 1993.

[24] S. Nielsen, personal communication, June 18, 1996.

[25] Y. Pant, B. Henderson-Sellers, and J.M. Verner, "Generalization of Object-Oriented Components for Reuse: Measurement of Effort and Size Change," *J. Object-Oriented Programming*, vol. 9, pp. 19-41, 1996.

[26] R.S. Pindyck and D.L. Rubinfeld, *Econometric Models & Economic Forecasts*. New York: McGraw-Hill, 1991.

[27] H. Schaefer, "Metrics for Optimal Maintenance Management," *Proc. Conf. Software Maintenance*, pp. 114-119, Washington, 1985,

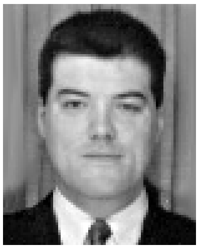
[28] R.C. Sharble and S.S. Cohen, "The Object-Oriented Brewery: A Comparison of Two Object-Oriented Development Methods," *Software Eng. Notes*, vol. 18, pp. 60-73, 1993.

[29] M. Shepperd, "A Critique of Cyclomatic Complexity as a Software Metric," *Software Eng. J.*, vol. 3, pp. 30-36, 1988.

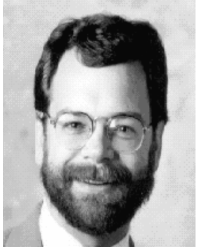
[30] D.P. Tegarden, S.D. Sheetz, and D.E. Monarchi, "Effectiveness of Traditional Software Metrics for Object-Oriented Systems," *Proc. Conf. Systems Science*, Hawaii, 1992,

[31] H. White, "A Heteroskedasticity–Consistent Covariance Matrix Estimator and a Direct Test for Heteroskedasticity," *Econometrica*, vol. 48, pp. 817-838, 1980.

Shyam R. Chidamber holds a BSEE degree from Bombay University, India; an MSEE degree from the University of Pittsburgh; and a PhD degree from the Massachusetts Institute of Technology. Dr. Chidamber is currently manager of the Working Council of Chief Information Officers at The Corporate Advisory Board in Washington DC. He has published articles in *IEEE Transactions on Software Engineering*, the *Journal of Software and Systems*, and the *International Journal of Technology Management*. Prior to his PhD studies, he was a member of the technical staff at AT&T Bell Laboratories in New Jersey.



David P. Darcy holds a bachelor of commerce degree and a masters of management science degree from the University College Dublin, Ireland, where he was also a member of the faculty. Darcy is a doctoral candidate majoring in information systems at the Katz Graduate School of Business, University of Pittsburgh.



Chris F. Kemerer received his BS degree from the Wharton School at the University of Pennsylvania and his PhD degree from Carnegie Mellon University. Dr. Kemerer is the David M. Roderick chair in information systems at the University of Pittsburgh. Prior to accepting his current position, he was an associate professor at MIT's Sloan School of Management. His research interests include management and measurement issues in information systems and software engineering.