An Empirical Study on Object-Oriented Metrics

Mei-Huei Tang Ming-Hung Kao Mei-Hwa Chen
Computer Science Department
SUNY at Albany
Albany, NY 12222
(meitang, kao, mhc)@cs.albany.edu

Abstract

The objective of this study is the investigation of the correlation between object-oriented design metrics and the likelihood of the occurrence of object-oriented faults. Such a relationship, if identified, can be utilized to select effective testing techniques that take the characteristics of the program under test into account. Our empirical study was conducted on three industrial real-time systems that contain a number of natural faults reported for the past three years. The faults found in these three systems are classified into three types: object-oriented faults, object management faults and traditional faults. The object-oriented design metrics suite proposed by Chidamber and Kemerer is validated using these faults. Moreover, we propose a set of new metrics that can serve as an indicator of how strongly object-oriented a program is, so that the decision to adopt object-oriented testing techniques can be made, to achieve more reliable testing and yet minimize redundant testing efforts.

1. Introduction

Object-oriented programming has many useful features, such as information hiding, encapsulation, inheritance, polymorphism and dynamic binding. These object-oriented features facilitate software reuse and component-based development. However, they might cause some types of faults that are difficult to detect using traditional testing techniques. For example, if a fault in an inherited function is encountered only in the context of the derived class, then this fault cannot be detected without the selected testing technique forcing an invocation of this function in an object which binds to this derived class. Our previous study [10] suggests that traditional testing techniques, such as functional testing, statement testing and branch testing, are not viable for detecting OO faults. To overcome these deficiencies, it is necessary to adopt an object-oriented testing technique that takes these features into account. However, the extent to which the cost and benefit we can balance by adopting an object-oriented testing depends on how the program under test has been implemented. We observe that it is not unusual for an object-oriented program to be syntactically ported from a traditional non-OO program or implemented using very few object-oriented features. Under such circumstances, the program is very unlikely to have object-oriented faults; therefore, additional effort in conducting object-oriented testing is not necessary.

Performing effective testing under cost and schedule constraints relies heavily on the selection of testing techniques, while the selection decision must be based on the characteristics of the program. Software metrics are the quantitative measurement of the complexity of the software or its design; therefore, they are good candidates for guiding the selection of testing techniques. Object-oriented metrics have been studied and proposed as good predictors for fault-prone modules/classes, for program maintainability and for software productivity. Our empirical study is aimed at identifying object-oriented metrics that can be utilized to characterize the degree of object-orientation an object-oriented program contains, so that the likelihood of object-oriented faults occurring can be estimated. This study, conducted on three industrial real-time programs, has two parts; the first part of the study is the validation of CK metrics on these programs. Then, through the observations obtained from the first part of the study, in the second part we identify a set of new metrics that might better serve our needs.

The remainder of the paper is organized as follows: Section 2 gives an overview of the existing studies in object-oriented metrics. The application programs used in this study and the analysis of the faults found in these programs are described in Sections 3 and 4. The study of CK metrics as well as the new metrics we proposed is presented in Section 5. Our conclusions, from this empirical study, and future research directions are given in Section 6.

2. Related Work

Chidamber and Kemerer [4] proposed a suite of object-oriented design metrics which were developed based on the ontology of Bunge. They analytically evaluated the metrics against Weyuker's measurement theory principles [18] and provided an empirical sample of these metrics from two commercial systems. Several studies have been conducted to validate CK metrics. Basili, Briand and Melo [1] presented the results of an empirical validation of CK metrics. Based on eight medium sized school projects they applied a logistic regression model to investigate whether these metrics can be used as fault-prone class indicators. Their results suggest that five of the six CK metrics are useful quality indicators for predicting fault-prone classes. Li and Henry [14] used two size metrics and eight OO metrics, including five of CK metrics, to empirically validate the applicability of these metrics on the number of lines changed per class, assumed to be related to maintenance effort. This empirical validation was conducted on two commercial systems using multiple linear regression technique. Their results show that OO metrics can be used to predict maintenance effort, measured by the number of lines changed per class, in an object-oriented system. Li [13] also theoretically validated CK metrics using metric-evaluation framework proposed by Kitchenham et. al. [12]. He discovered some deficiencies of CK metrics in the evaluation process and proposed a new suite of OO metrics that overcome these deficiencies. Chidamber et. al. [3] further explored the applicability of CK metrics on practical managerial work such as productivity and rework effort. Their empirical results suggest CK metrics were significant economic variable indicators for the three commercial OO systems used in their study.

Moreover, other object-oriented metrics are proposed to complement CK metrics. In [2], a new suite of coupling measures for object-oriented design was proposed and empirically validated using logistic regression technique. They discovered that not all of the import and export coupling measures are significant predictors of class fault-proneness. Their data also suggest that these OO coupling measurement metrics are complementary quality indicators to CK OO metrics. In [7], a set of object-oriented design metrics, MOOD metrics, was proposed and theoretically validated from measurement point of view. They also empirically validated this set of metrics and the results suggest that MOOD metrics operate at the system level is complementary to CK's class level OO metrics.

To use software metrics in guiding testing resources allocation, Harrison [8] evaluated several traditional testing resources allocation techniques such as resource allocation by module size and by complexity. He used McCabes cyclomatic complexity measurement, Halsteads effort measure and Harrison and Cooks MMC metric for resource allocation by complexity. He concluded that none of the resource allocation techniques were perfect and more work need to be done in this area.

3. Descriptions of the Applications

The applications used in this empirical study are subsystems of an HMI (Human Machine Interface) software which is a fully networked Supervisory Control and Data Acquisition system. This software is based on client-server architecture consisting of servers and clients. Servers are responsible for the collection and distribution of data. Clients connect into servers and have full access to the collected data for viewing and control actions. This software, which consists of more than 200 subsystems and 3 million lines of code, has been used by many manufacturing companies for several years.

Although each subsystem selected plays a different role in the system and performs a different functionality, they share some similar characteristics that meet with our selection criteria. All the subsystems are implemented using Microsoft Visual C++^{TM} under the Windows NT^{TM} environment and possess certain object-oriented features such as encapsulation, inheritance and polymorphism. These subsystems are briefly described as below.

System A is a user interface-oriented program that allows customers to configure the basic product operations and device communications. It consists of 20 classes that define 256 new, re-defined or virtual functions, and approximately 5,600 lines of code in length. System B is a real time data logging process that collects data as needed and logs data into the database, based on the user configuration. This system defines 45 classes and 353 new, re-defined or virtual functions, comprising approximately 21,300 lines of code. System C is a communication-oriented program that acts as a router not only delivering messages between processes within the same host but also forwarding messages to other hosts. This system defines 27 classes and 293 new, re-defined or virtual functions and contains approximately 16,000 lines of code. Among these three systems, Systems A and B were designed and implemented in a certain object-oriented methodology, whereas System C was ported from a program written in C programming language and partially redesigned and enhanced by its engineers to support the new requirements of a new operational environment.

4. Fault Analysis

We analyzed the trouble reports of the three systems recorded for the past three years and classified the faults found in the system test and maintenance phases. The

classification scheme was based on the nature of these faults and their relevance to the object-orientation. Below we describe the detailed classification of these types among which *Type I* and *Type II* are OO faults and *Type III* faults are non-OO.

Type I: Object-oriented faults that are strongly related to the OO features and are introduced by these features such as inheritance and polymorphism. This type of faults can be further divided into two sub-categories: inheritance faults and polymorphism faults. A typical inheritance fault occurs when a derived class modifies a data member of the base class, which in turn changes the behavior of the base class and then causes the fault. In other words, the derived class changes the environment of the base class which causes the faults encountered in either the derived class or the base class. Polymorphism faults are the faults encountered in the OO program when an object can be bound to different classes during the runtime. For instance, if there are X possible bindings of an object which sends a service request and Y possible bindings of the other object which provides the service, then totally there are X x Y different possible combinations of bindings during the runtime. If some of them are not tested during a system test phase, then a failure, caused by a polymorphism fault, might occur.

Type II: The object management faults that are related to object management such as object copying, dangling reference, object memory usage faults and so on. A typical object copying fault would be encountered if the implementation of the method for copying an object is either a duplication of the original object or the generation of a reference to the original object. If the copied method is used incorrectly, some unexpected faults or memory corruption will occur. The dangling reference object fault happens when an object, say "object A," tries to reference another object, "object B," which was destroyed by a third object, "object C." The object memory usage fault normally refers to the situation where an object allocates memory during runtime and does not free up this resources when it is no longer needed.

Type III: The traditional types of faults that are not related to objects. They fall into the fault classification of traditional software [16].

In Table 1 we summarize the results obtained from the analysis of these three systems. The upper half of the table lists the system size in thousands of lines of code and the number of classes defined in such system. The total number of known faults and their distributions in the three fault types is presented along with the percentages of *type I* and *type II* faults to the total known faults. We observe that among the total faults found, one third of the faults in Systems A and B are OO faults, and in System C these faults types comprise about one fifth of the total faults.

Table 1. Summary of the faults in the three systems.

System	A	B	C
Lines of code	5.6k	21.3k	16.0k
Number of classes	20	45	27
Number of faults	35	80	85
Type I faults	5	15	10
Type II faults	6	13	7
Type III faults	24	52	68
OO faults(%)	31%	35%	20%

5. Object-Oriented Metrics

It is often desirable that the fault-prone modules and the types of residential faults can be estimated based on some quantitative measurement of a given system. Object-oriented metrics are developed to realize the structure and the characteristic of object-oriented programs. Some metrics, such as CK metrics [4], have been proven empirically to be useful for the prediction of fault-prone modules [1].

In this study, we measured the CK metrics of the three systems, described in Section 3, and analyzed their distributions in these systems.

5.1 CK metrics

The object-oriented metrics proposed by Chidamber and Kemerer [4] are described as follows:

Weighted methods per class (WMC): This measures the complexity of an individual class. Two different weighting functions are considered: WMC1 uses the nominal weight of 1 for each function, and hence measures the number of functions. WMC uses a weighting function which is 1 for functions accessible to other modules and 0 for private functions. In this study, we adopted the first approach to simplify the factors. In another word, we consider all methods of a class to be equally complex.

Depth of inheritance tree of a class (DIT): It is defined as the length of the longest path of inheritance ending at the current module. Intuitively, the deeper the inheritance tree for a class, the harder it might be to predict its behavior due to the interaction between the inherited features and new features.

Number of children (NOC): It represents the number of classes that inherit directly from the current class. Moderate values for this measure indicate the scope for reuse; however, high values may indicate an inappropriate abstraction in the design. Furthermore, a class with a large number of children has to provide more generic service to all the children in various contexts and must be more flexible. We believe that this tends to introduce more complexity into this parent class.

Coupling between objects (CBO): This provides the number of other modules that are coupled to the current module either as a client or a supplier. A class is coupled to another if it uses the member functions and/or instance variables of the other class. Excessive coupling indicates weakness of module encapsulation and may inhibit reuse. The assumption behind this metric is that highly coupled classes tend to introduce more faults caused by inter-class activities.

Response for a class (RFC): This gives the number of methods that can potentially be executed in response to a message received by an object of that class. The larger the number of methods that could potentially respond to a message, the greater the complexity of that class.

5.2. Statistical Analysis

In order to investigate the correlation between OO metrics and fault-prone (including traditional and OO faults) classes, we conducted logistic regression analysis, a standard classification technique [9] based on maximum log likelihood estimation, to analyze the relationships between explanatory independent variables and binary dependent variables. In our study we used univariate logistic regression to evaluate the relationships between individual OO metric and fault-prone classes.

A logistic regression model can be defined as

$$Prob(X_1, X_2, ..., X_n) = \frac{\exp^{(B_0+B_1*X_1+...+B_n*X_n)}}{1+\exp^{(B_0+B_1*X_1+...+B_n*X_n)}}$$

where X_i , i= 1, 2, .., n, are the explanatory independent variables (OO metrics), exp is the base of the natural logarithms, approximately 2.718 and $Prob$ is the probability of detection of a specific type of faults in a class. An univariate logistic regression model is a special case of the above formula with only one variable in the formula.

$$Prob(X) = \frac{\exp^{(B_0+B_1*X)}}{1+\exp^{(B_0+B_1*X)}}$$

Table 2 and 4 show the results obtained through univariate logistic regression on system A, B and C. Only the metrics that are significant are included in these tables and for each metric the following statistics are reported:

Coefficients (B_i's): the estimated logistic regression coefficients are estimated from data through maximization of the log likelihood function, and measure the respective independent variable's (metric's) contribution in the dependent variable. The larger the absolute coefficient values, the larger (positive or negative according to the sign) the impact of the explanatory variables (OO metrics) on the probability of fault detection.

The statistical significance (p-value): represents the degree of accuracy of coefficient estimation. More specifically, the p-value represents the probability of error that is involved in accepting observed results as valid. The larger the statistical significance, the less believable the estimated impact of the explanatory independent variable (OO metric). In our study we used 0.1 as the significance threshold.

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

$$R^2 = \frac{LL_0 - LL}{LL_0}$$

where LL_0 is the log likelihood of the data under null hypothesis without any variable and LL is the log likelihood of the data under the model.

Odds ratio: represents the change in odds when the value of an independent variable increases by 1. The odds of an event occurring is defined as the ratio of the probability of having a fault over the probability of not having a fault. Odds ratio is provided as an evaluation of the impact of explanatory independent variables (OO metrics) on the dependent variable.

5.3. Analysis of CK Metrics

Table 2 shows the descriptive statistics of CK metrics for three systems. Table 3 shows the results of our logistic regression analysis on CK metrics, where WMC/RFC(all) denotes the analysis of WMC/RFC w.r.t. the classes that contain faults regardless of type; while WMC/RFC(oo) indicates the analysis of WMC/RFC w.r.t. the classes that contain OO faults. Among the five evaluated CK metrics, only WMC and RFC are shown to be significant (with p-value $<$ 0.1) indicators for OO faults and the total number of faults in both System A and B. For system C, WMC is a significant fault-prone class detector while RFC is a very significant OO fault indicator. In sum, for all three systems, WMC

Table 2. Descriptive statistics of CK metrics.

SYSTEM A					
	WMC	DIT	NOC	CBO	RFC
Minimum	1	0	0	1	0
Maximum	37	3	2	5	291
Median	10.5	1	0	1	42.5
Mean	11.85	1.25	0.2	1.65	56.55
Std. Dev.	8.51	0.71	0.52	1.03	64.09
SYSTEM B					
Metrics	WMC	DIT	NOC	CBO	RFC
Minimum	1	0	0	0	0
Maximum	27	5	17	28	205
Median	5	2	0	2	12
Mean	6.81	1.54	0.70	3.29	27.16
Std. Dev.	5.73	1.27	2.63	4.72	40.34
SYSTEM C					
Metrics	WMC	DIT	NOC	CBO	RFC
Minimum	2	0	0	0	0
Maximum	89	2	2	5	397
Median	7	1	0	1	3
Mean	10.37	0.89	0.24	1.17	35.75
Std. Dev.	15.66	0.55	0.63	1.33	76.73

Table 3. Summary of CK metrics.

SYSTEM A				
Metrics	Coefficient	p-value	R^2	Odds Ratio
WMC(all)	0.2298	0.054	0.2438	1.2583
RFC(all)	0.0546	0.0401	0.3135	1.0562
WMC(oo)	0.1652	0.07	0.207	1.1796
RFC(oo)	0.0439	0.0441	0.3170	1.0449
SYSTEM B				
Metrics	Coefficient	p-value	R^2	Odds Ratio
WMC(all)	0.3728	0.0219	0.1831	1.4517
RFC(all)	0.0634	0.0499	0.1666	1.0654
WMC(oo)	0.1897	0.0254	0.1207	1.2089
RFC(oo)	0.0411	0.0218	0.1704	1.0419
SYSTEM C				
Metrics	Coefficient	p-value	R^2	Odds Ratio
WMC(all)	0.2305	0.0602	0.1417	1.2592
RFC(oo)	0.0579	0.0033	0.4999	1.0596

Table 4. Observations on significant CK metrics.

	WMC	RFC
Type A	28%	29%
Type B	36%	36%
Type OO	50%	42%

is a significant fault-prone class indicator and RFC is a significant OO fault-prone class indicator. On the other hand, although the logistic regression results show WMC and RFC are significant indicators, our observations show that there are 36% of faulty classes that have less than average WMC value might be overlooked if we only use WMC for testing resources allocation and this observation also holds for the RFC. Moreover 50%, and 42% of classes that have OO faults, but with less than average WMC and RFC values, might be overlooked if we only consider CK metrics. These results are summarized in Table 4 where *Type A* is the percentage of faulty classes which have high WMC/RFC values over the total number of classes in all three systems; *Type B* is the percentage of faulty classes which have low WMC/RFC values and *Type OO* is the percentage of classes which have OO faults yet have low WMC/RFC values. Thus additional metrics might be needed to provide a better fault-prone class prediction.

The scenario described above can be explained by the following observations:

1. The complexity of the method is not considered. For example, a method with 1000 lines of code is likely to introduce more faults than a method with 100 lines of code.

2. The dynamic behavior is not considered. For example, a class which is used more frequently than other classes tends to have more reported faults than other classes.

3. The number of child classes, including those who inherit directly and indirectly from the current module, should be considered. The existing NOC metric reflects only the number of direct descendants for each class. Therefore, the additional complexity introduced by indirect descendants is not considered.

4. The function dependency relationship between the inherited methods and the new/redefined methods in child classes. A method is *function dependent* on another method if the former method uses any data which is defined/modified by the latter method. The motivation behind this factor is that when a data member, which is used by the inherited methods, is modified by the new or redefined method, new faults tend to be introduced into the inherited methods.

5. A class with more object/memory allocating activities tends to introduce more *type II* faults. Therefore, the number of object/memory allocating statements within a class should be taken into account. Furthermore, a complicated copy/assign operator of a class tends to introduce more *type II* faults than the default or simple one. Therefore, the complexity of the copy/assign operator implementation of a class should be also considered.

In summarizing from this study, we observe that CK metrics may not be sufficient for identifying fault-prone classes with OO faults. For this purpose, other metrics are needed which take into account (1)the dynamic behavior of the program; and (2) the scenarios that the instances of the classes are referenced in the program.

5.4. New Metrics

In this section we present a set of new metrics which are derived from our observations in studying CK metrics.

Inheritance Coupling: (IC) The IC provides the number of parent classes to which a given class is coupled. A class is coupled to its parent class if one of its inherited methods is functionally dependent on the new or redefined methods in the class. In general, a class is coupled to its parent class if one of the following conditions holds:

1. One of its inherited methods uses a variable (or data member) that is defined in a new/redefined method.
2. One of its inherited methods calls a redefined method and uses the return value of the redefined method.
3. One of its inherited methods is called by a redefined method and uses a parameter that is defined in the redefined method.
4. One of its inherited methods uses a variable X, and the value of X depends on the value of a variable Y which is defined in a new/redefined method.

The motivation behind the IC metric is that when a data member, which is used by an inherited method, is modified by a new or redefined method, it is likely to introduce new faults into the inherited method.

Coupling Between Methods: (CBM) The CBM provides the total number of new/redefined methods to which all the inherited methods are coupled. An inherited method is coupled to a new/redefined method if it is functionally dependent on a new/redefined method in the class. Therefore, the number of new/redefined methods to which an inherited method is coupled can be measured.

The CBM measures the total number of function dependency relationships between the inherited methods and new/redefined methods. As a matter of fact, this metric is a variant of the IC metric. The motivation behind this metric is that the IC only measures the number of parent classes to which a given class is coupled, without the CBM, additional function dependency complexity at the methods level is not considered.

Number of Object/Memory Allocation: (NOMA) It measures the total number of statements that allocate new objects or memories in a class. The indirect allocations, ie. the allocations caused by calling other methods, are not considered. The motivation behind this metric is that classes with large numbers of object/memory allocation statements tend to introduce additional complexity for object/memory management. Therefore, the higher the NOMA, the higher the probability of detecting object management faults.

Average Method Complexity: (AMC) The AMC provides the average method size for each class. Pure virtual methods and inherited methods are not counted. The assumption behind this metric is that a large method, which contains more code, tends to introduce more faults than a small method.

5.5. Analysis of the New Metrics

Table 5 shows the results of the logistic regression analysis on the new metrics where (all) refers to all types of faults and (oo) refers to OO faults only. AMC is a significant fault-prone class predictor and NOMA is a significant OO fault indicator for both system A and B. IC, CBM and NOMA are significant fault-prone class indicators for system B. In Table 6 we summarized the percentage of classes where *Type A* denotes the faulty classes that have high IC/CBM/AMC, *Type B* denotes the faulty classes that have low IC/CBM/AMC and *Type OO* denotes the classes that have low IC/CBM/AMC while still containing OO faults. Although the logistic regression results show that IC, CBM and AMC were significant OO fault indicators for all three systems, our observations show that there are 30%, 35% and 23% of faulty classes with less than average IC, CBM and AMC values which might be overlooked. Also there are 24%, 34% and 26% of classes with OO faults, having less than average IC, CBM and AMC values which might be overlooked if we only consider these metrics. Thus the use of CK metrics with our additional four new metrics to

Table 5. Summary of the new metrics.

SYSTEM A				
Metrics	Coefficient	p-value	R^2	Odds Ratio
AMC(all)	0.4183	0.036	0.5924	1.5194
IC(oo)	3.4965	0.0084	0.3867	32.9984
CBM(oo)	2.6156	0.0197	0.5554	13.6756
NOMA(oo)	1.9542	0.0262	0.2789	7.0585
AMC(oo)	0.2264	0.0558	0.3580	1.254
SYSTEM B				
Metrics	Coefficient	p-value	R^2	Odds Ratio
IC(all)	2.1162	0.01	0.1582	8.2997
CBM(all)	1.4466	0.029	0.2171	4.2468
NOMA(all)	0.6954	0.0439	0.0955	2.0046
AMC(all)	0.1621	0.0012	0.4421	1.176
IC(oo)	2.9299	0.0001	0.3174	18.725
CBM(oo)	1.6595	0.0024	0.3594	5.2569
NOMA(oo)	0.7199	0.0162	0.1243	2.0542
AMC(oo)	0.0699	0.0017	0.1981	1.0724
SYSTEM C				
Metrics	Coefficient	p-value	R^2	Odds Ratio
IC(oo)	2.2069	0.0167	0.2203	9.0872
CBM(oo)	0.5659	0.0343	0.2165	1.761
AMC(oo)	0.0513	0.0069	0.3559	1.0526

Table 6. Observations on the significant new metrics.

	IC	CBM	AMC
Type A	35%	30%	42%
Type B	30%	35%	23%
Type OO	24%	34%	26%

achieve a higher percentage of fault detection becomes an important issue for testing resources allocation.

We observe that by using WMC as the only fault prone class indicator for testing resources allocation, we might miss 36% of faulty classes and 50% of OO fault prone classes, which have less than average WMC values. If we use CBM or AMC by itself, 35% and 23% of faulty classes and 34% and 26% of OO fault classes might be missed, respectively. However, if we use WMC as faulty class indicator and use traditional testing techniques on these faulty classes and then use CBM and AMC as OO fault class indicators and use OO testing techniques on the classes that have either high CBM or AMC values, we will be able to discover 91.22% of OO faults and 90% of total faults. Thus we can conclude that after using WMC as a traditional testing resources allocation indicator, if there are classes that do not test with either high CBM or AMC values, it is necessary to apply OO testing techniques to these classes, in order to achieve a high percentage of fault detection.

6. Conclusions and Future Work

We have validated CK metrics using three industrial real-time systems and the results suggest that WMC can be a good indicator for faulty classes and RFC is a good indicator for OO faults. Furthermore, we present a set of new metrics which we consider useful as indicators of OO fault-prone classes. Therefore, these new metrics can be utilized to decide which classes need to be tested using OO testing techniques. From the observations of this study, we also suggest how to use these metrics effectively.

Our future research direction aims at studying how to systematically implement these metrics to guide the selection/prioritization of testing techniques. Therefore, not only identifying which metrics are good indicators of fault-prone classes but also describing how to apply these metrics in the testing process.

References

[1] V. R. Basili, L. C. Briand, and W. L. Melo. A validation of object-oriented design metrics as quality indicators. *IEEE Transactions on Software Engineering*, 22(10):751–761, October 1996.

[2] L. Briand, P. Devanbu, and W. Melo. An investigation into coupling measures for c++. Technical Report ISERN-96-08, ISERN, 1996.

[3] S. R. Chidamber, D. P. Darcy, and C. F. Kemerer. Managerial use of metrics for object-oriented software: An exploratory analysis. *IEEE Transactions on Software Engineering*, 24(8):629–639, August 1998.

[4] S. R. Chidamber and C. F. Kemerer. A metrics suite for object oriented design. *IEEE Transactions on Software Engineering*, 20(6):476–493, June 1994.

[5] S. D. Conte, H. E. Dunsmore, and U. Y. Shen. *Software Engineering Metrics and Models*. The Benjamin/Cummings Publishing Company, Inc., 1986.

[6] N. E. Fenton. *Software Metrics: A Rigorous Approach*. Chapman & Hall, 1991.

[7] R. Harrison and S. J. Counsell. An evaluation of the mood set of object-oriented software metrics. *IEEE Transactions on Software Engineering*, 21(12):929–944, December 1995.

[8] W. Harrison. Using software metrics to allocate testing resources. *Journal of Management Information Systems*, 4(4):93–105, 1988.

[9] D. W. Hosmer and S. Lemeshow. *Applied Logistic Regression*. John Wiley & Sons, 1989.

[10] M. Kao, M. H. Tang, and M. H. Chen. Investigating test effectiveness on object-oriented software - a case study. In *Proceedings of Twelfth Annual International Software Quality Week*, 1999.

[11] B. Kitchenham. *Software Metrics: Measurement for Software Process Improvement*. BlackwellPublishers Inc., 1996.

[12] B. Kitchenham, S. L. Pfleeger, and N. Fenton. Towards a framework for software measurement validation. *IEEE Transactions on Software Engineering*, 21(12):929–944, December 1995.

[13] W. Li. Another metric suite for object-oriented programming. *Journal of Systems and Software*, 44:155–162, 1998.

[14] W. Li and S. Henry. Object-oriented metrics that predict maintainability. *Journal of Systems and Software*, 23:111–122, 1993.

[15] M. Lorenz and J. Kidd. *Object-Oriented Software Metrics*. Prentice Hall Inc., 1994.

[16] I. C. Society. Ieee 1044 - standard classification for software errors, faults and failures. *IEEE Computer Society*, 1994.

[17] B. F. Webster. *Pitfalls of Object-Oriented Development*. M&T Books, New York, 1995.

[18] E. Weyuker. Evaluating software complexity measures. *IEEE Transactions on Software Engineering*, 14:1357–1365, 1988.