Empirical Validation of Three Software Metrics Suites to Predict Fault-Proneness of Object-Oriented Classes Developed Using Highly Iterative or Agile Software Development Processes

Hector M. Olague, Letha H. Etzkorn, *Senior Member*, *IEEE*, Sampson Gholston, and Stephen Quattlebaum

Abstract—Empirical validation of software metrics suites to predict fault proneness in object-oriented (OO) components is essential to ensure their practical use in industrial settings. In this paper, we empirically validate three OO metrics suites for their ability to predict software quality in terms of fault-proneness: the Chidamber and Kemerer (CK) metrics, Abreu's Metrics for Object-Oriented Design (MOOD), and Bansiya and Davis' Quality Metrics for Object-Oriented Design (QMOOD). Some CK class metrics have previously been shown to be good predictors of initial OO software quality. However, the other two suites have not been heavily validated except by their original proposers. Here, we explore the ability of these three metrics suites to predict fault-prone classes using defect data for six versions of Rhino, an open-source implementation of JavaScript written in Java. We conclude that the CK and QMOOD suites contain similar components and produce statistical models that are effective in detecting error-prone classes. We also conclude that the class components in the MOOD metrics suite are not good class fault-proneness predictors. Analyzing multivariate binary logistic regression models across six Rhino versions indicates these models may be useful in assessing quality in OO classes produced using modern highly iterative or agile software development processes.

Index Terms—Object-oriented software metrics, object-oriented metrics, software quality metrics, software maintenance programming, software reuse.

1 Introduction

Many OO metrics have been developed by researchers to help assess design quality [1], [2], [3], [4], [5]. While a measure may be correct from a theoretical (measurement) perspective, it may not be of practical use in industrial settings. Metrics may be difficult to collect or may not really measure the intended quality properties of software. Empirical validation is necessary to demonstrate the usefulness of a metric in practical applications [6], [4]. El Emam et al. give a summary in [7] of 17 prior research studies undertaken to empirically validate the usefulness of OO metrics in identifying fault-prone classes; these include [6], [8], [9], [3], [10].

- *H.M. Olague is with the US Army Space and Missile Defense Command, SMDC-RDTI-S, PO Box 1500, Huntsville, AL 35807. E-mail: holague@cs.uah.edu.*
- *L.H. Etzkorn is with the Computer Science Department, University of Alabama in Huntsville, Huntsville, AL 35899. E-mail: letzkorn@cs.uah.edu.*
- *S. Gholston is with the Industrial and Systems Engineering and Engineering Management Department, University of Alabama in Huntsville, Huntsville, AL 35899. E-mail: gholston@ise.uah.*
- *S. Quattlebaum is with Dragonfly Athletics, 262 Peach Tree Road, Hartselle, AL 35640. E-mail: stephen@covidimus.net.*

Manuscript received 18 May 2006; revised 17 Nov. 2006; accepted 15 Mar. 2007; published online 28 Mar. 2007.
Recommended for acceptance by K. Kanoun.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number TSE-0110-0506.
Digital Object Identifier no. 10.1109/TSE.2007.1015.

Many of the published studies cited in El Emam et al. [7] emphasize validation of the CK metrics suite. Also mentioned in El Emam et al. [7] are four additional metrics that have been the subject of validation studies in the past: NMO (measuring the number of inherited methods overridden by a subclass), NMA (measuring the number of methods added by a subclass), SIX (a combination of inherited metrics), and NPAVG (measuring the average number of parameters per method). OO metrics validation studies using fault-proneness as a quality indicator that have appeared since the publication of [7] include [11], [12], [13], [14], [15].

However, using OO class metrics as quality predictors during the entire software development life cycle may be useful when a highly iterative or agile software process is employed. Highly iterative or agile software development processes depart from the idea of having all of the requirements defined prior to the initiation of coding activities. That is, new requirements are constantly being introduced and realized during each new iteration. For example, some agile methods, such as Timeboxing, use very short four-week to eight-week software development iterations (each iteration includes requirements analysis, implementation, testing, and delivery), where new requirements are solicited from the customers on each iteration. If improvements are continuously being made in spiral development fashion, the continued use of OO class metrics may be appropriate since established components will undergo changes and new

components will be added to support new requirements. This will likely be a source of new defects.

In this research, we performed a case study, which conducted yet another empirical validation of OO class metrics. However, while most other studies have focused on the analysis on a single version, or release, of a software system, we focused on empirical validation of the software produced by a highly iterative software development process that can be considered an agile software process. The two questions we wished to answer are 1) can the OO metrics suites we employed identify fault-prone classes in software developed using a highly iterative, or agile, development process during its initial delivery, and 2) can the OO metrics suites we employed identify fault-prone classes in multiple, sequential releases of software developed using a highly iterative or agile process? Our test case was multiple versions of a robust, publicly available open-source software product, Rhino [16].

For the Rhino software we analyzed for this case study, the software development strategy employed is highly iterative, with a bottom-up approach. According to Norris Boyd, creator of the Mozilla Rhino project [17], Rhino may be considered an example of the use of the agile software development model in open source software [17]. In Rhino, new software enhancements usually are followed by new software defects due to the code integration and testing philosophy employed by agile systems (and Rhino) in general [17]. The development of Rhino meets all but one of the 12 principles defined by the Agile Alliance [18] for those who want to achieve agility, as they would occur in open source software. Specifically how Rhino meets the 12 principles is discussed further in the background section (see Section 2.2).

In addition to the widely validated CK metrics suite, we empirically validated two additional OO metrics suites: Brito e Abreu's Metrics for OO Design (MOOD) [19], and Bansiya and Davis' Quality Metrics for OO Design (QMOOD) [1]. Both metrics suites have been implemented in commercial and publicly available OO metrics tool suites [20], [21]. We chose these two metrics suites because not many previous statistical analyses have been performed on them, to our knowledge.

In this study, since it has previously been shown that some of the CK metrics are good predictors of initial quality [7], we compared individual MOOD and QMOOD metrics to the CK metrics to see if they are better at predicting initial quality in OO classes. We examined the metrics from all three metrics suites together to determine whether they measure different dimensions of OO class quality or are measuring the same thing. Then, we developed models using the different metric suites to predict faults.

This paper is organized as follows: In Section 2, we provide background for understanding the paper. This includes definitions of the metrics we examined, as well as a description of the software data we examined and the tools we used to collect the metrics. In Section 3, we describe our analysis methods and the experiments we performed. Section 4 contains the results of our experiments. In particular, Section 4.3.2 contains the models we developed using multivariate binary logistic regression for each of the metrics suites examined. The validation of the models is also shown in Section 4.3.2. A discussion of our results is in Section 4.4. Conclusions are in Section 5, and future research is in Section 6.

2 Background

2.1 Metrics Used in This Study

Three different metrics suites were used in this paper: Chidamber and Kemerer's metrics suite [3], the MOOD metrics suite [2], and the QMOOD metrics suite [1].

2.1.1 Chidamber and Kemerer's (CK) Metrics

Chidamber and Kemerer originally defined the CK metrics suite in 1991. In 1994, they published another paper containing revised definitions of some of the metrics [3]. The metrics implemented for this research are from the 1994 paper except where otherwise noted: For example, the LCOM metric is a later variation of the original CK LCOM that has been shown by other researchers to be a better metric than the 1994 LCOM [4], [22]. The CK metrics are described in Table 1.

2.1.2 Brito e Abreu's MOOD Metrics

Brito e Abreu developed a small suite of metrics that could be used to evaluate systems designed with an "outside-in" methodology whereby early preparation and planning for development are considered of special importance in order to save money in the long run [10]. This emphasis on early planning led to metrics that do not depend to a great extent on the definitions of functions, so they can be collected early in the design phase. He felt metrics should be easy to compute, so as not to overwhelm those tasked with evaluating a system and they should have an unambiguous formal definition that was not tied to any particular OO language, should use consistent units, and should result in numbers independent of the system size. He accomplished the latter of these goals by formulating all of his metrics to result in a probability value between 0 and 1. In later versions, Brito e Abreu introduced the concept of language "bindings," conventions for describing software systems in any language in the terminology used in MOOD [10]. The MOOD metrics used in this research are described in Table 1. Harrison et al. conducted a theoretical evaluation of the MOOD metrics suite in [22] and concluded that the metrics could be used to provide an overall assessment of a software system. They also call for empirical validation of the metrics suite, which is one of the subgoals of this research [22].

2.1.3 Bansiya and Davis's Quality Model for Object Oriented Design (QMOOD) Metrics

The QMOOD metrics are defined so that metrics calculated on a system can be used to compute a kind of supermetric, the Total Quality Index (TQI). Like the MOOD metrics, the QMOOD metrics are defined to be computable early in the design process. Bansiya and Davis first decided on a set of design quality attributes, based loosely on the attributes defined in the International Standards Organization (ISO) 9126 standard: reusability, flexibility, understandability, functionality, extendibility, and effectiveness. Then, they identified a set of object-oriented design properties that support the design quality attributes. There are 11 of these design properties; for each one, Bansiya and Davis identified a metric that captures the design property. The QMOOD class metrics analyzed in this study are described in Table 1.

2.2 Software Data Examined

We chose the Mozilla Rhino project to examine in this study because it was a real project (not a contrived software

TABLE 1
Metrics Definitions

Chidamber and Kemerer's (CK) Metric Suite (Class Metrics Only) [3]	
Metric Name Value	**Definition**
Weighted Methods Per Class (WMC)	*Sum of complexities of local methods of a class. For simple WMC, when all complexities are unity, same as number of class methods.*
Depth of Inheritance Tree (DIT)	*Max number of edges between a given class and a root class in an inheritance graph (0 for a class which has no base classes).*
Num.Children(NOC)	*A count of the number of direct children of a given class.*
Coupling Between Objects (CBO)	*Counts other classes whose attributes or methods are used by the given class plus those that use the attributes or methods of the given class.*
Response For a Class (RFC)	*A count of all of local methods of a class plus all of methods on other classes directly called by any of the methods on the class.*
Lack of Cohesion of Methods (LCOM)	*Num. of disjoint sets of local methods, no two sets intersect, any two methods on same set share at least one local variable* (1998 definition).
Fernando Brito e Abreu's MOOD Metric Suite (Class Metrics Only) [2]	
Attribute Hiding Factor (AHF)	*[1-total num. visible (can be accessed)) attributes in a set of classes] / total num. attributes in the set. Measures visibility of a class definition.*
Method Hiding Factor (MHF)	*[1-total num. visible (can be called) methods in a set of classes] / total num. methods in the set. Measures visibility of a class definition.*
Attribute Inheritance Factor (AIF)	*The ratio of inherited attributes to the total number of attributes in a class.*
Method Inheritance Factor (MIF)	*The ratio of inherited methods to the total number of methods in a class.*
Bansiya and Davis' QMOOD Metric Suite (Class Metrics Only) [1]	
Avg. Num. Ancestors (QMOOD_ANA)	*Average of DIT for all classes in the system.*
Cohesion Among Methods (QMOOD_CAM)	*A measure of cohesion that is based on the similarity of method signatures in a class. Included for completeness; not implemented in this research.*
Class Interface Size (QMOOD_CIS)	*The count of public methods in a class.*
Data Access Metric (QMOOD_DAM)	*The ratio of private or protected attributes to the total number of attributes declared in a class.*
Direct Class Coupling (QMOOD_DCC)	*A count of classes that accept instances of a given class as a parameter plus classes including attributes of the given class' type.*
Measure of Aggregation (QMOOD_MOA)	*The percentage of data declarations in the system whose types are of user defined classes, as opposed to those of system defined classes such as integers, real numbers, etc.*
Measure of Fnctnl. Abstraction (QMOOD_MFA)	*Same as MOOD_MIF.*
Number of Methods (QMOOD_NOM)	*The number of methods in a class. Same as WMC when weights of the methods in the class equal unity.*

project for the study) and because of the availability of fault data for several versions of the project. Rhino is an open-source implementation of JavaScript. The Rhino software is completely written in Java. It is typically embedded in Java applications to provide scripting to end users. Rhino was begun several years ago by Netscape, to produce a Java version of the popular Navigator Internet browser. The Rhino software development team consists of three

TABLE 2
Rhino Descriptive Statistics

		ck-cbo	ck-dit	ck-lcom98	ck-noc	ck-rfc	ck-wmc	wmc-mc	mood-ahf	mood-aif	mood-mhf	mood-mif	qmood-cis	qmood-dam	qmood-dcc	qmood-mfa	qmood-nom
RHINO 14R3	**# Class**	95	95	95	95	95	95	95	95	95	95	95	95	95	95	95	95
	# Zeros	35	65	0	85	3	3	3	84	66	18	67	18	4	52	73	3
	Mean	2.33	.41	2.13	.32	13.59	12.17	41.80	.07	.24	.68	.21	9.04	.93	1.95	.21	12.17
	STDEV	3.58	.69	2.23	1.65	18.28	16.69	72.93	.22	.39	.40	.35	13.81	.21	5.91	.35	16.69
Per-centiles	**25**	.00	.00	1.00	.00	2.00	2.00	3.00	.00	.00	.33	.00	1.00	1.00	.00	.00	2.00
	50	1.00	.00	2.00	.00	7.00	6.00	9.00	.00	.00	.90	.00	3.00	1.00	.00	.00	6.00
	75	3.00	1.00	2.00	.00	16.00	13.00	38.00	.00	.65	1.00	.49	11.00	1.00	2.00	.49	13.00
RHINO 15R1	**# Class**	126	126	126	126	126	126	126	126	126	126	126	126	126	126	126	126
	# Zeros	42	86	0	108	5	5	5	119	92	40	98	24	6	76	98	5
	Mean	2.78	.42	2.22	.32	14.22	12.35	47.43	.06	.24	.64	.20	8.45	.94	1.94	.20	12.35
	STDEV	4.34	.70	2.38	1.55	20.10	17.66	86.35	.20	.39	.40	.34	13.12	.20	6.61	.34	17.66
RHINO 15R2	**# Class**	179	179	179	179	179	179	179	179	179	179	179	179	179	179	179	179
	# Zeros	70	135	80	158	5	5	5	167	144	54	143	29	8	120	143	5
	Mean	2.23	.49	2.42	.25	13.36	11.81	43.64	.05	.17	.62	.17	7.03	.95	1.69	.17	11.81
	STDEV	3.93	1.05	2.27	1.08	18.61	16.34	80.30	.19	.33	.39	.33	11.31	.19	7.04	.33	16.34
RHINO 15R3	**# Class**	178	178	178	178	178	178	178	178	178	178	178	178	178	178	178	178
	# Zeros	67	134	0	157	5	5	5	166	143	53	142	27	8	120	142	5
	Mean	2.28	.49	2.44	.25	13.48	11.99	44.33	.05	.17	.61	.17	7.03	.95	1.69	.17	11.99
	STDEV	4.05	1.05	2.27	1.09	19.15	17.00	81.60	.19	.33	.38	.33	11.65	.19	7.07	.33	17.00
RHINO 15R4	**# Class**	198	198	198	198	198	198	198	198	198	198	198	198	198	198	198	198
	# Zeros	68	150	0	175	3	3	3	188	160	66	159	37	8	131	159	3
	Mean	2.42	.47	2.37	.24	13.24	11.67	42.84	.05	.17	.57	.17	6.32	.95	1.71	.17	11.67
	STDEV	4.19	1.01	2.12	1.07	19.59	17.07	80.13	.18	.33	.39	.33	11.34	.18	7.49	.33	17.07
RHINO 15R5	**# Class**	201	201	201	201	201	201	201	201	201	201	201	201	201	201	201	201
	# Zeros	72	151	0	179	10	10	10	180	160	181	154	41	10	135	154	10
	Mean	2.25	.49	2.47	.25	13.28	11.86	44.09	.06	.18	.54	.20	6.33	.94	1.61	.20	11.86
	STDEV	3.86	.99	2.27	1.13	19.21	17.02	83.48	.20	.34	.38	.36	11.64	.20	6.92	.36	17.02
Per-centiles	**25**	.00	.00	1.00	.00	3.00	2.00	3.00	.00	.00	.13	.00	1.00	1.00	.00	.00	2.00
	50	1.00	.00	2.00	.00	6.00	5.00	11.00	.00	.00	.61	.00	3.00	1.00	.00	.00	5.00
	75	3.00	.50	3.00	.00	16.00	14.00	40.00	.00	.00	.93	.03	7.00	1.00	1.00	.03	14.00

programmers, all in separate locations worldwide and delivering working software with a varying cycle time (from 2 to 16 months). More history can be found at [16]. Rhino versions 14R3, 15R1, 15R2, 15R3, 15R4, and 15R5 were analyzed in this study. Error data exists for Rhino in the online Bugzilla repository [23]. In our Rhino fault data collection, performed under a NASA grant, the change logs for each version of Rhino were examined, which listed the bugs that were resolved for that version of Rhino. Bug fixes were cross-referenced with classes affected by each bug/fix. Descriptive statistics for the Rhino software versions we used in this analysis are contained in Table 2. Figs. 1 and 2 provide information about the number of classes, defects, and enhancements found in each version of Rhino.

The development of Rhino meets all but one of the 12 principles defined by the Agile Alliance [18] for those who want to achieve agility, as they would occur in open source software. The 12 principles are as follows:

1. Early and continuous delivery of software.
2. Welcome changing requirements, even late in development.
3. Deliver working software frequently.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals.
6. The most efficient and effective method to convey information is face-to-face conversation.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. Sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective and adjusts its behavior accordingly.

Rhino's agile development cycle is largely due to the incorporation of software extensions and improvements by generous third parties [17]. This meets principles 1, 2, and 3, principle 4 (in open source software, the users/clients of the software are typically the developers), principle 5 (again, the users are often the developers in open source software), principle 7, principle 8 (since users provide updates continually as the software is used), and principle 10 (simplicity—the users typically would only perform improvements necessary to their own work). Principles 11 and 12 are covered in that the users who need the updates are the teams. The Rhino team performs regression testing on Rhino prior to releasing a distribution but relies heavily on user feedback to try out a new version and report problems they encounter [17], which meets principle 9 (continued attention to technical excellence and good design). The only

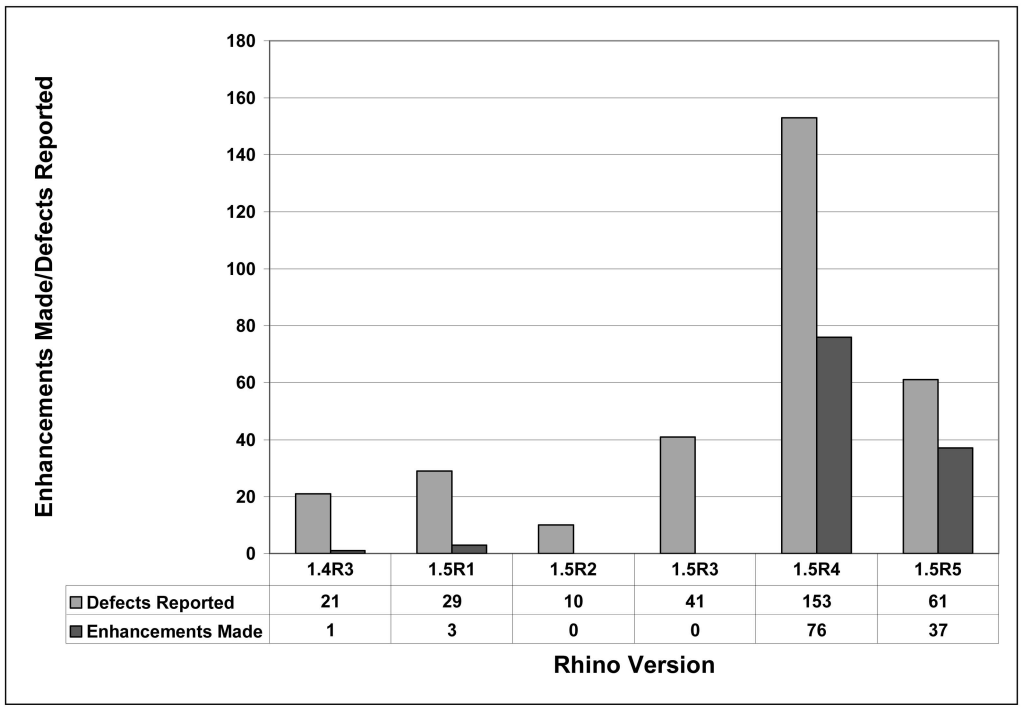


Fig. 1. Defects reported and enhancements made per Rhino version.

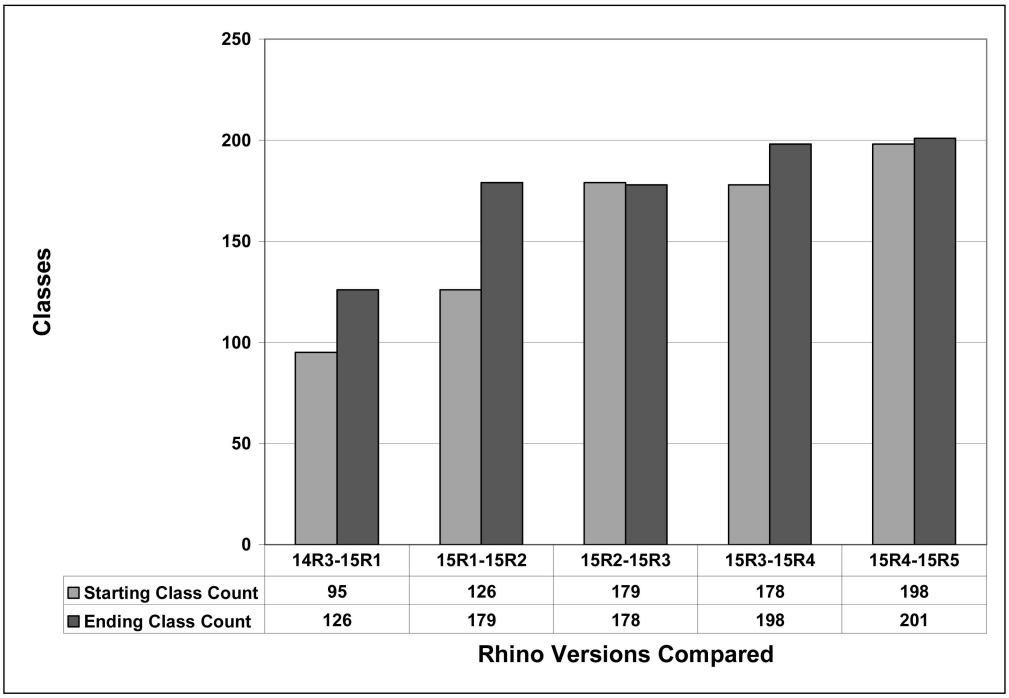


Fig. 2. Class growth comparison between different Rhino versions.

principle that is not usually met is that of face-to-face conversation, principle 6.

2.3 Software Tool Used

Several commercial tools implement a subset of the metrics targeted in this study. However, no commercial tool currently supports them all for Java, the programming language used by the Rhino project. Therefore, we used the Software System Markup Language (SSML) tool chain, a metrics collection tool [20]. The SSML tool chain is a set of tools that work together to translate source code into an intermediate source-language-independent representation and then perform analysis on this representation. One of the tools in the SSML tool chain yielded the metrics used in this paper.

3 Analysis Methods Employed

Our case study examining the three metrics suites (CK, MOOD, and QMOOD) was performed over six releases of the Rhino software. Further description of the Rhino

software is provided in Section 2.2. After fault data was collected and analyzed for Rhino, we used SSML tools to collect OO metrics from the source code for each version of Rhino (see Section 2.3). We collected metrics from all the classes in each version of Rhino. From the variation in the number of classes between Rhino versions (see Fig. 2), the degree to which additional requirements are being added to Rhino from version to version is indicated.

In our case study, we first want to know whether the metrics from the three different suites are related to each other, that is, whether they are measuring different dimensions of OO class quality or whether they measure the same thing. To do this, we first analyzed intercorrelations between the CK class metrics only, collected on the Rhino data, and compared our results to those of other researchers (see Section 4.1). This was intended as a calibration step of sorts, to provide an initial indication as to whether any potential major differences could be expected between our case study and previous case studies.

Second, in order to determine which metrics can be used individually as fault predictors, we performed a bivariate correlation between defects and the individual metrics from the three metrics suites. This is described in Section 4.2. Similar studies have been performed on the CK metrics in the past [7], but little such work has been done for the QMOOD and MOOD metrics suites. We used only Rhino versions 15R3, 15R4, and 15R5 because the number of faults took on a variety of values, unlike earlier versions, which took on values of zero or one only.

Finally, we developed models using the different metrics suites to predict faults. In previous studies, the lack of variability in the response variable has been the principal motivation to forgo the use of traditional linear regression techniques in favor of logistic regression. However, we examined the distribution of the number of defects found in classes of all six versions of the Rhino software and concluded that there was good variability in the response (dependent) variable in three later versions of Rhino, although this good variability was not present in the other versions of Rhino. Thus, we developed multivariate linear regression (ordinary least squares, or OLS) models (using the normal distribution) for Rhino versions 15R3, 15R4, and 15R5 using the best subsets, backward, and forward regression techniques. However, this yielded poor models that explained, on average, less than 35 percent of the variability in the data. Therefore, these models are not included in this paper.

Since the models developed using OLS techniques were unsuccessful, we employed binary logistic regression analysis to develop models to predict faults. As mentioned before, this was necessary for some versions of Rhino due to the lack of variability in the response variable. Logistic regression does not assume linearity of relationship between the independent and dependent variables. It also does not require normally distributed variables and does not assume that the variance around the regression line is the same for all values of the predictor value (homoscedasticity). Additionally, binary logistic regression has been shown to provide good models for fault proneness prediction in the past [7]. We refer the reader to Hosmer and Lemeshow [24] for more details on binary logistic regression and its application.

First, we performed univariate binary logistic regression (UBLR) of metrics versus faults (Section 4.3.1) to determine which variables (OO class metrics) were statistically significant quality indicators. Next, we performed a collinearity analysis (Section 4.3.2) to determine which variables to include in the multivariate binary logistic regression (MBLR) models. Finally, we developed the models (Section 4.3.2). Using Rhino version 14R3, we developed three models for the CK metrics, two models for the MOOD metrics, and two models for the QMOOD metrics. In developing our models, if the class had one or more faults, we classified it as fault-prone ($\text{fault-prone} = 1$); otherwise, it was classified as fault-free ($\text{fault-prone} = 0$). This assessment was done on each version of Rhino. We validated our models using a simple holdout method (Section 4.3.2), where we developed MBLR models with Rhino version n data and validated them with Rhino version $n+1$ data, for versions 14R3 through 15R5. To control for size, we examined model performance separately using all classes, small classes (simple $\text{WMC} \leq 10$), and large classes (simple $\text{WMC} > 10$).

3.1 Goal Statement and Research Hypothesis

Two research hypotheses will be tested under the common goal:

Goal. *To assess the ability of OO metrics to identify fault-prone components in different software development environments.*

Hypothesis 1. *OO metrics can identify fault-prone classes in traditional and highly iterative or agile developed OO software during its initial delivery (initial quality).*

Hypothesis 2. *OO metrics can identify fault-prone classes in multiple, sequential releases of OO software systems developed using highly iterative or agile software development processes.*

4 Results

4.1 Comparison of CK Interrelationships in Rhino to Previous Statistical Studies

In our first analysis of this section, we compared intercorrelations between the CK metrics collected on the Rhino data to similar correlations performed by previous researchers. In the past, Basili et al. [6], Briand et al. [13], and, more recently, Ferenc et al. [15], conducted correlations between the CK metrics and showed varying degrees of correlation between the CK class metrics. Table 3 compares correlations from the Rhino data to the results of previous studies by others using the correlation coefficient and the Hopkins criteria (< 0.1 trivial, 0.1-0.3 minor, 0.3-0.5 moderate, 0.5-0.7 large, 0.7-0.9 very large, and 0.9-1 almost perfect) [25]. We used the Spearman Rank correlation in our analysis due to the nonparametric nature of the OO class metrics in this study.

Our correlation study results indicate CK-WMC correlates more strongly with *CK-RFC*, *CK-CBO*, and *CK-LCOM* than in previous studies. In our study, DIT had minor to moderate correlation with *CK-RFC*, *CK-CBO*, or *CK-LCOM*, whereas some previous studies showed a small degree of correlation. Rhino's *CK-RFC* metric correlated more strongly with *CK-CBO* and *CK-LCOM* than occurred in previous studies. The primary differences between this study and previous studies were *CK-NOC* in this study had either no correlation or minor correlation with *CK-CBO* (previous studies showed no correlation) and *CK-CBO* had a moderate to large correlation with *CK-LCOM* (previous studies showed a small or no correlation).

TABLE 3
Six Versions of Rhino Spearman CK Metrics Bivariate Intercorrelations Compared to Ranges from Published Studies

		ck wmc	ck dit	ck rfc	ck noc	ck cbo	ck lcom98
ck-wmc	**Range***	1	≤S	S-L	Ø	S-M	**M-L**
	RHINO	1	S-M	P	S	L	**L-VL**
ck dit	**Range***		1	Ø -L	Ø	Ø-S	**T**
	RHINO		1	S-M	Ø-S	S-M	**S-M**
ck rfc	**Range***			1	Ø	M	**T-M**
	RHINO			1	S	L	**L-VL**
ck noc	**Range***				1	Ø	**Ø**
	RHINO				1	Ø-M	**Ø-S**
ck cbo	**Range***					1	**≤S**
	RHINO					1	**M-L**
ck lcom98	**Range***						**1**
	RHINO						**1**

*** Range of correlation over several published studies.**
Legend: Ø = None; T = Trivial (<0.10); S = Minor (0.11-0.30); M = Moderate (0.31-0.50); L = Large (0.51-0.70); VL = Very Large (0.71 – 0.90); P = Perfect (0.91-1.00)

TABLE 4
Spearman Defects Correlations for Three Versions of Rhino

Rhino Version	Stat.	ck cbo	ck dit	ck lcom98	ck noc	ck rfc	ck wmc	mood ahf	mood aif	mood mhf	mood mif	qmood cis	qmood dam	qmood dcc	qmood mfa	qmood nom
15R3	Coeff.	0.216**	0.122	0.331**	0.203**	0.373**	0.381**	0.264**	0.088	-0.090	0.109	0.247**	-0.278**	0.074	0.109	0.381**
(N=178)	Sig.	0.004	0.105	0.000	0.007	0.000	0.000	0.000	0.244	0.235	0.149	0.001	0.000	0.328	0.149	0.000
15R4	Coeff.	0.470**	0.384**	0.503**	0.346**	0.551**	0.529**	0.271**	0.343**	-0.003	0.364**	0.423**	-0.201**	0.094	0.364**	0.529**
(N=198)	Sig.	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.965	0.000	0.000	0.005	0.187	0.000	0.000
15R5	Coeff.	0.178*	-0.037	0.218**	0.198**	0.250**	0.236**	0.206**	-0.041	-0.216**	-0.054	0.015	-0.226**	0.101	-0.054	0.236**
(N=201)	Sig.	0.012	0.603	0.002	0.005	0.000	0.001	0.003	0.562	0.002	0.449	0.832	0.001	0.153	0.449	0.001

** Correlation is significant at the 0.01 level (2-tailed).* Correlation is significant at the 0.05 level (2-tailed).

4.2 Bivariate Correlation between Defects and CK, MOOD, and QMOOD Class Metrics Components

In order to determine which metrics can be used individually as fault predictors, we performed a bivariate Spearman correlation between defects and the individual metrics from the three metrics suites. Similar studies have been performed on the CK metrics in the past [15], but little such work has been done for the QMOOD and MOOD metrics suites. We used only Rhino versions 15R3, 15R4, and 15R5 because the number of faults took on a variety of values, unlike earlier versions, which took on values of zero or one only.

Again using the Hopkins scale, we determine that *CK-RFC* (S-L) and *CK-WMC* (S-L) correlate most strongly with class defects and, therefore, can be used individually as predictors of fault-prone classes. These metrics provided large to very large (S-L) statistically significant correlation. *QMOOD-DAM* provided consistent minor (S) positive correlation. Finally, *CK-CBO* and *CK-LCOM* provided small to minor to moderate (S-M) correlation. *CK-NOC* provided the least amount of consistent, statistically significant correlation (S-M) with number of defects. The results are contained in Table 4.

4.3 Logistic Regression Analysis

We performed binary logistic regression analysis on the different Rhino versions. In Section 4.3.1, we describe a univariate binary logistic regression (UBLR) of the metrics versus faults to determine which metrics were statistically significant indicators of quality. In Section 4.3.2, we describe how we used multivariate binary logistic regression (MBLR) to construct predictive models for the CK suite and the QMOOD suite. We also developed MOOD models, but they did not perform as well as the CK and QMOOD models.

4.3.1 Univariate Binary Logistic Regression for CK, MOOD, and QMOOD Class Metrics

In this section, we performed a univariate binary logistic regression (UBLR) of the metrics versus faults to determine which metrics were statistically significant indicators of quality. The results of the UBLR are shown in Tables 5 and 6. The UBLR analysis for the CK metrics reveals that the CK-CBO coefficient is significant at the $\alpha = .05$ level for five of the six versions of Rhino we examined (14R3, 15R1, 15R3, 15R4, and 15R5). The log likelihood (LL) test statistic G (which tests the null hypothesis that all the coefficients associated with the predictors equal zero versus these coefficients not all being equal to zero) indicates that there is sufficient evidence that at least one of the coefficients is different from zero since its p-value is below our critical value of 0.025 in all five cases. The inferential goodness-of-fit test that we used was the Hosmer-Lemeshow (HL) test [24]. It showed a good fit of the data for three of the five versions of Rhino ($\text{p-value} > 0.05$). The other three measures of association used, the Somer's D, Goodman-Kruskal Gamma, and Kendall Tau-a, also showed a reasonable fit of the data to the model; however, we do not show these due to space limitations. Similar analysis shows the CK-LCOM98 coefficient was significant in four

TABLE 6
Univariate Logistic Regression—CK, MOOD, QMOOD—Rhino 15R3 through Rhino 15R5

Rhino 15R3	ck cbo	ck dit	ck lcom98	ck noc	ck rfc	ck wmc	ck wmcM	mood ahf	mood aif	mood mhf	mood mif	qmood cis	qmood dam	qmood dcc	qmood mfa	qmood nom
Const	-2.59	-2.12	-3.10	-2.13	-3.42	-3.39	-3.19	-2.15	-2.06	-1.68	-2.13	-2.64	-0.44	-2.08	-2.13	-3.39
Coeff	0.19	0.19	0.37	0.31	0.07	0.08	0.02	1.74	0.28	-.0.575	0.59	0.07	-1.71	0.03	0.59	0.08
pvalue	0.00	0.31	0.00	0.05	0.00	0.00	0.00	0.05	0.68	0.34	0.37	0.00	0.05	0.00	0.37	0.000
odds ratio	1.21	1.21	1.44	1.36	1.08	1.08	1.02	5.68	1.32	0.56	1.80	1.07	0.18	1.03	1.80	1.08
LL	-57.8	-64.1	-56.6	-62.6	-46.6	-47.0	-44.5	-63.0	-64.5	-64.1	-64.2	-56.8	-63.0	-63.8	-64.2	-47.0
G	13.55	0.92	15.92	3.98	36.00	35.24	40.28	3.29	0.17	0.91	0.78	15.54	3.18	1.51	0.78	35.24
pvalue	0.00	0.34	0.00	0.05	0.00	0.00	0.00	0.07	0.69	0.34	0.38	0.00	0.07	0.22	0.38	0.000
HL	3.74	2.68	8.32	1.18	13.66	6.92	4.32	7.47	9.55	19.63	3.77	8.61	5.79	3.92	3.77	6.92
	4	2	3	1	8	8	8	1	2	5	2	7	1	2	2	8
	0.44	0.26	0.04	0.28	0.09	0.55	0.83	0.01	0.01	0.00	0.15	0.28	0.02	0.14	0.15	0.55
Rhino 15R4	**ck cbo**	**ck dit**	**ck lcom98**	**ck noc**	**ck rfc**	**ck wmc**	**ck wmcM**	**mood ahf**	**mood aif**	**mood mhf**	**mood mif**	**qmood cis**	**qmood dam**	**qmood dcc**	**qmood mfa**	**qmood nom**
Const	-1.58	-1.37	-2.42	-1.06	-2.08	-2.04	-1.76	-1.02	-1.39	-1.04	-1.46	-1.48	-0.25	-1.01	-1.46	-2.04
Coeff	0.23	0.70	0.57	0.31	0.08	0.09	0.02	0.80	2.04	0.10	2.28	0.08	-0.77	0.02	2.28	0.09
pvalue	0.00	0.00	0.00	0.10	0.00	0.00	0.00	0.33	0.00	0.81	0.00	0.00	0.35	0.38	0.00	0.00
odds ratio	1.26	2.02	1.77	1.37	1.08	1.09	1.02	2.23	7.68	1.11	9.75	1.08	0.46	1.02	9.75	1.09
LL	-104	-106	-97.5	-114	-91.8	-93.2	-93.31	-116	-106	-116	-104	-107	-115.6	-115.6	-103.6	-93.2
G	24.29	20.15	37.14	3.97	48.50	45.70	45.41	0.92	19.86	0.06	24.88	18.21	0.84	0.77	24.88	45.70
pvalue	0.00	0.00	0.00	0.05	0.00	0.00	0.00	0.34	0.00	0.81	0.00	0.00	0.36	0.38	0.00	0.00
HL	26.49	15.29	6.48	12.81	17.63	10.70	15.64	13.29	6.37	37.71	11.11	28.97	2.48	1.13	11.11	10.70
	4	2	3	1	7	7	8	1	2	5	2	6	0	2	2	7
	0.00	0.00	0.09	0.00	0.01	0.15	0.05	0.00	0.04	0.00	0.00	0.00	*	0.57	0.00	0.15
Rhino 15R5	**ck cbo**	**ck dit**	**ck lcom98**	**ck noc**	**ck rfc**	**ck wmc**	**ck wmcM**	**mood ahf**	**mood aif**	**mood mhf**	**mood mif**	**qmood cis**	**qmood dam**	**qmood dcc**	**qmood mfa**	**qmood nom**
Const	-2.34	-1.80	-2.64	-2.03	-2.74	-2.71	-2.48	-2.01	-1.77	-1.02	-1.78	-2.18	-0.28	-1.92	-1.78	-2.71
Coeff	0.16	-0.16	0.27	0.45	0.05	0.05	0.01	1.73	-0.58	-1.90	-0.47	0.04	-1.73	0.03	-0.47	0.05
pvalue	0.00	0.51	0.00	0.02	0.00	0.00	0.00	0.02	0.40	0.00	0.46	0.01	0.02	0.18	0.64	0.00
odds ratio	1.18	0.85	1.31	1.56	1.05	1.06	1.01	5.61	0.56	0.15	0.62	1.04	0.18	1.03	0.62	1.06
LL	-72.8	-79.1	-73.4	-75.2	-66.5	-67.2	-68.0	-77.0	-78.9	-73.6	-79.0	-76.2	-77.0	-78.5	-79.0	-67.2
G	12.92	0.48	11.76	8.27	25.60	24.17	22.57	4.59	0.77	11.46	0.58	8.18	4.60	1.59	0.58	24.17
pvalue	0.00	0.49	0.00	0.00	0.00	0.00	0.00	0.03	0.38	0.00	0.45	0.00	0.03	0.21	0.45	0.00
HL	8.99	0.09	2.90	1.93	10.63	11.80	9.49	0.50	4.17	8.05	0.57	9.43	0.74	8.19	0.57	11.80
	4	1	3	1	7	6	8	0	1	5	1	5	0	2	1	6
	0.06	0.77	0.41	0.17	0.16	0.07	0.30	*	0.04	0.15	0.45	0.09	*	0.02	0.45	0.07

of the six versions of Rhino we attempted to fit. The HL test shows a good fit of the data in three of those four cases (Rhino versions 15R2, 15R4 and 15R5). For HL, the χ^2, degrees of freedom, and p-value are also given.

The CK-RFC coefficient was significant in all six versions of Rhino we analyzed and the data fit was good in four of the six versions. The CK-WMC coefficient was significant in four of the six versions of Rhino and the data fit was good in four of the six versions. The MOOD metric coefficients were significant in only two of the six Rhino versions. The QMOOD-CIS coefficient was significant in five of the six versions of Rhino and a good data fit in five of the six. The QMOOD-CIS metric is similar to the CK-WMC metric and produced similar results.

4.3.2 Multivariate Binary Logistic Regression

In this section, we perform a collinearity analysis to determine which models to use in our multivariate binary logistic regression (MBLR) models. Then, we develop MBLR models separately for the three different suites of metrics, CK, QMOOD, and MOOD. Finally, we validate our models using a simple holdout method.

Collinearity analysis and MBLR model variable selection. In selecting variables for a multivariate binary logistic regression, we first look for potential collinearity problems in the bivariate correlations between all the metrics within their respective metrics suite. Tables 7, 8, and 9 show these correlations. Clearly, there are large, very large, and perfect correlations between two variables in every one of the metrics suites. These indicate a potential collinearity problem when using combinations of these variables in a multivariate regression analysis. However, since every model we propose to build includes more than two regressors, we will need to examine the *variance inflation factor* (VIF) of all the potential regressors in each model. Therefore, all of the class metrics for each metric suite will be examined together to determine which metrics are appropriate to combine in a regression analysis model.

The VIF and the condition number will be used in our collinearity analysis and model variable selecton. The VIF is the reciprocal of the *tolerance*. The tolerance is an indicator of the variance percentage that cannot be accounted for by other regressors (independent variables in the regression equation). The VIF is always greater than or equal to 1 and is the "number of times the variance of the corresponding parameter estimate is increased due to multicollinearity as compared to as it would be if there was no multicollinearity." There is no "industry standard" VIF threshold value established. In general, in OLS regression, a variable whose VIF value is greater than 10 may indicate possible multicollinearity problems and should be investigated further [26]. In logistic regression, values of VIF above 2.5 may require further investigation. We will use a VIF threshold value of 3.0 and an objective value of 2.5 for evaluating potential regressors for multicollinearity problems [26]. If

TABLE 5
Univariate Logistic Regression—CK, MOOD, QMOOD—Rhino 14R3 through Rhino 15R2

Rhino 14R3	**Ck cbo**	**ck dit**	**ck lcom98**	**ck noc**	**ck rfc**	**ck wmc**	**ck wmcM**	**mood ahf**	**mood aif**	**mood mhf**	**mood mif**	**qmood cis**	**qmood dam**	**qmood dcc**	**qmood mfa**	**qmood nom**
Const	-2.19	-1.94	-2.11	-1.86	-2.69	-2.46	-2.63	-1.73	-1.96	-1.08	-1.98	-2.11	-1.10	-1.61	-1.98	-2.46
Coeff	0.18	0.54	0.19	0.54	0.06	0.05	0.02	0.69	0.97	-0.95	1.19	0.04	-0.62	-0.04	1.19	0.05
pvalue	0.01	0.13	0.13	0.13	0.00	0.00	0.00	0.55	0.15	0.15	0.11	0.03	0.59	0.61	0.11	0.00
odds ratio	1.19	1.71	1.20	1.72	1.06	1.05	1.02	1.98	2.63	0.39	3.29	1.04	0.54	0.96	3.29	1.05
LL	-37.9	-40.4	-39.9	-38.5	-33.4	-35.7	-31.9	-41.3	-40.4	-40.4	-40.2	-39.0	-41.3	-41.3	-40.2	-35.7
G	6.93	2.16	3.02	5.92	15.96	11.45	19.10	0.33	1.99	2.20	3.51	4.98	0.27	0.37	2.51	11.45
pvalue	0.01	0.14	0.08	0.02	0.00	0.01	0.00	0.57	0.16	0.16	0.11	1.07	0.60	0.54	0.11	0.00
HL	7.02	0.84	6.35	0.23	11.59	14.41	17.43	2.01	4.59	24.42	1.69	8.03	1.88	0.98	1.69	14.41
	4	1	3	1	8	7	8	1	2	4	2	7	1	1	2	7
	0.14	0.36	0.10	0.63	0.17	0.04	0.03	0.16	0.10	0.00	0.43	0.33	0.17	0.32	0.43	0.04
Rhino 15R1	**Ck cbo**	**ck dit**	**ck Lcom98**	**ck noc**	**ck rfc**	**ck wmc**	**ck wmcM**	**mood ahf**	**mood aif**	**mood mhf**	**mood mif**	**qmood cis**	**qmood dam**	**qmood dcc**	**qmood mfa**	**qmood nom**
Const	-2.26	-2.63	-2.10	-1.81	-2.67	-0.27	-2.74	-1.84	-2.34	-1.96	-2.34	-2.32	-0.57	-1.76	-2.34	-2.66
Coeff	0.15	0.93	0.15	0.20	0.051	0.06	0.02	1.27	1.81	0.34	2.12	0.06	-1.26	0.02	2.12	0.06
pvalue	0.01	0.00	0.97	0.16	0.00	0.00	0.00	0.19	0.00	0.59	0.00	0.00	0.20	0.63	0.00	0.001
odds ratio	1.16	2.53	1.17	1.22	1.05	1.06	1.01	3.57	6.13	1.41	8.29	1.06	0.28	1.02	8.29	1.06
LL	-48.9	-49.1	-51.9	-52.2	-43.8	-43.6	-39.9	-52.7	-48.8	-53.3	-48.4	-47.8	-52.7	-53.3	-48.4	-43.6
G	9.05	8.72	3.13	2.48	19.26	19.73	27.00	1.51	9.23	0.29	10.07	11.20	1.48	0.21	10.07	19.73
pvalue	0.00	0.00	0.08	0.12	0.00	0.00	0.00	0.22	0.00	0.59	0.00	0.00	0.22	0.65	0.00	0.000
HL	11.44	7.72	9.60	0.42	9.89	3.85	4.44	2.75	9.45	14.65	9.353	9.27	4.66	4.03	9.35	3.85
	5	1	3	1	8	8	8	1	2	4	3	7	1	3	3	8
	0.04	0.01	0.02	0.52	0.27	0.87	0.82	0.10	0.01	0.01	0.03	0.23	0.03	0.26	0.03	0.87
Rhino 15R2	**Ck cbo**	**ck dit**	**ck lcom98**	**ck noc**	**ck rfc**	**ck wmc**	**ck wmcM**	**mood ahf**	**mood aif**	**mood mhf**	**mood mif**	**qmood cis**	**qmood dam**	**qmood dcc**	**qmood mfa**	**qmood nom**
Const	-3.42	-3.37	-3.94	-3.21	-4.07	-4.00	-4.34	-3.14	-3.27	-2.20	-3.41	-3.39	-4.78	-3.15	-3.41	-4.00
Coeff	0.07	0.26	0.23	0.03	0.04	0.04	0.01	-1.67	0.35	-2.15	0.95	0.02	1.64	-0.04	0.95	0.04
pvalue	0.20	0.36	0.02	0.92	0.00	0.00	0.00	0.65	0.75	0.05	0.35	0.29	0.65	0.77	0.35	0.00
odds ratio	1.08	1.29	1.26	1.03	1.04	1.04	1.01	0.19	1.42	0.12	2.58	1.02	5.13	1.26	2.58	1.04
LL	-28.9	-29.2	-27.1	-29.6	-24.7	-25.3	-21.5	-29.4	-29.5	-27.3	-29.2	-29.1	-29.4	-29.5	-29.2	-25.3
G	1.25	0.72	4.92	0.01	9.81	8.613	16.18	0.32	0.10	4.56	0.81	0.84	0.31	0.17	0.81	8.61
pvalue	0.26	0.40	0.03	0.92	0.00	0.00	0.00	0.57	0.75	0.03	0.37	0.36	0.58	0.68	0.37	0.00
HL	6.66	2.62	2.89	2.72	23.74	22.70	5.56	0.34	8.55	4.38	13.55	5.63	0.35	0.87	13.55	22.70
	4	2	3	1	8	8	8	0	2	4	2	7	0	1	2	8
	0.15	0.27	0.41	0.10	0.00	0.00	0.70	*	0.01	0.36	0.00	0.58	*	0.35	0.00	0.00

TABLE 7
Range of CK Spearman Bivariate Correlations over All Versions of Rhino

	CK-CBO	CK-DIT	CK-LCOM98	CK-NOC	CK-RFC	CK-WMC	CK-WMC-McCABE
CK-CBO	**1.0**	S-M	M-L	Ø-S	L	L	L
CK-DIT		**1.0**	S-M	Ø-M	S-M	S-M	S-M
CK-LCOM98			**1.0**	S	L-VL	L-VL	L-VL
CK-NOC				**1.0**	Ø-M	Ø-M	Ø-M
CK-RFC					**1.0**	P	P
CK-WMC						**1.0**	P
CK-WMC-McCABE							**1.0**

Legend:
Ø = None; T = Trivial (<0.10); S = Minor (0.11-0.30); M = Moderate (0.31-0.50); L = Large (0.51-0.70); VL = Very Large (0.71 – 0.90); P = Perfect (0.91-1.00)

TABLE 8
Range of MOOD Spearman Bivariate Correlations over All Versions of Rhino

	MOOD-AHF	MOOD-AIF	MOOD-MHF	MOOD-MIF
MOOD-AHF	**1.0**	Ø	Ø	Ø
MOOD-AIF		**1.0**	Ø	P
MOOD-MHF			**1.0**	Ø-M
MOOD-MIF				**1.0**

Legend:
Ø = None; T = Trivial (<0.10); S = Minor (0.11-0.30); M = Moderate (0.31-0.50); L = Large (0.51-0.70); VL = Very Large (0.71 – 0.90); P = Perfect (0.91-1.00)

TABLE 9
Range of QMOOD Spearman Bivariate Correlation over All Versions of Rhino

	QMOOD-CIS	QMOOD-DAM	QMOOD-DCC	QMOOD-MFA	QMOOD-NOM
QMOOD-CIS	**1.0**	S-M	Ø-S	S-M	VL
QMOOD-DAM		**1.0**	Ø-S	Ø	S-M
QMOOD-DCC			**1.0**	Ø-S	Ø-S
QMOOD-MFA				**1.0**	S-M
QMOOD-NOM					**1.0**

Legend:
Ø = None; T = Trivial (<0.10); S = Minor (0.11-0.30); M = Moderate (0.31-0.50); L = Large (0.51-0.70); VL = Very Large (0.71 – 0.90); P = Perfect (0.91-1.00)

TABLE 10
Collinearity Analysis for CK Metrics

	VIF (usimg all CK class metrics)					
	14R3	**15R1**	**15R2**	**15R3**	**15R4**	**15R5**
CK-CBO	2.564	2.970	2.517	2.549	2.677	2.677
CK-DIT	1.014	1.026	1.080	1.084	1.065	1.039
CK-LCOM98	2.842	2.544	2.751	2.686	2.840	2.519
CK-NOC	1.432	1.411	1.164	1.149	1.158	1.257
CK-RFC	62.016	50.796	49.727	52.719	54.954	57.994
CK-WMC	53.414	37.588	40.700	44.340	45.952	52.492
CK-WMC-McCabe	6.268	6.884	5.232	5.307	5.558	5.980
Condition Number	29.379	25.284	25.960	26.648	26.734	27.912
	VIF (removing CK-WMC-McCabe and CK-RFC)					
	14R3	**15R1**	**15R2**	**15R3**	**15R4**	**15R5**
CK-CBO	2.431	2.838	2.175	2.173	2.174	2.235
CK-DIT	1.010	1.026	1.079	1.083	1.063	1.031
CK-LCOM98	2.565	2.366	2.655	2.613	2.779	2.369
CK-NOC	1.356	1.335	1.154	1.143	1.150	1.238
CK-RFC	**n/a**	**n/a**	**n/a**	**n/a**	**n/a**	**n/a**
CK-WMC	2.094	1.940	2.618	2.579	2.799	2.733
CK-WMC-McCabe	n/a	n/a	n/a	n/a	n/a	n/a
Condition Number	5.006	4.958	5.332	5.301	5.628	5.170
	VIF (removing CK-WMC and CK-RFC)					
	14R3	**15R1**	**15R2**	**15R3**	**15R4**	**15R5**
CK-CBO	2.405	2.717	1.980	1.957	1.978	1.991
CK-DIT	1.011	1.023	1.080	1.083	1.063	1.026
CK-LCOM98	2.063	2.142	2.228	2.274	2.288	1.910
CK-NOC	1.377	1.343	1.157	1.147	1.149	1.229
CK-RFC	**n/a**	**n/a**	**n/a**	**n/a**	**n/a**	**n/a**
CK-WMC	**n/a**	**n/a**	**n/a**	**n/a**	**n/a**	**n/a**
CK-WMC-McCabe	1.458	1.402	1.593	1.599	1.646	1.566
Condition Number	4.638	4.846	4.750	4.824	4.922	4.456
	VIF (removing CK-WMC and CK-WMC-McCabe)					
	14R3	**15R1**	**15R2**	**15R3**	**15R4**	**15R5**
CK-CBO	2.469	2.885	2.237	2.223	2.272	2.342
CK-DIT	1.010	1.025	1.079	1.083	1.063	1.031
CK-LCOM98	2.480	2.318	2.471	2.467	2.618	2.251
CK-NOC	1.357	1.333	1.154	1.143	1.149	1.233
CK-RFC	2.093	1.945	2.481	2.468	2.767	2.733
CK-WMC	**n/a**	**n/a**	**n/a**	**n/a**	**n/a**	**n/a**
CK-WMC-McCabe	**n/a**	**n/a**	**n/a**	**n/a**	**n/a**	**n/a**
Condition Number	4.931	4.959	5.095	5.113	5.443	5.034

the VIF values indicate potential multicollinearity problems, we will remove one or more variables and reevaluate the new potential model variable configuration. We will continue with this analysis until all model variables have VIF values less than 3.0. In addition, we will use the *condition number*, which is the ratio of the square root of the largest Eigenvalue to all the rest. Eigenvalues are those associated with the singular value decomposition of the variance-covariance matrix of the coefficients. In general, multicollinearity is likely to be a problem with a high condition number (more than 20 or 30) [26], [27]; low condition numbers indicate no multicollinearity problem.

Our VIF analysis for the CK metrics suite is contained in Table 10. The condition number is also given and is an indicator of collinearity problems. The first section shows the VIF analysis with all potential regressors. Collinearity problems are suspected with RFC, WMC, and WMC-McCabe since they exceed our VIF threshold value. After removing RFC and WMC-McCabe from the model, a subsequent VIF analysis shows that the remaining variables are within VIF threshold and objective values. Also, the condition number is significantly reduced. Another VIF analysis removing RFC and WMC shows similar results. Finally, the removal of WMC and WMC-McCabe from the model shows yet another configuration that may be successfully used. Based on these results, we will be developing and exploring the performance of three class metrics MBLR analysis model configurations of the CK metrics:

TABLE 11
Rhino Version 14R3 CK Multivariate Binary Logistic Regression Results

Multivariate Binary Logistic Regression Results for CK Metrics Suite, Model #1						
	Constant	**ck cbo**	**ck dit**	**ck lcom 98**	**ck noc**	**ckwmc**
Coeff	-2.87	0.06	0.65	-0.21	0.49	0.06
SE Coeff.	0.56	0.12	0.40	0.18	0.41	0.02
odds ratio	-	1.06	1.91	0.81	1.63	1.06
p-value	0.00	0.60	0.10	0.25	0.23	0.01
LL	-31.68, Tests that all slopes are zero: G= 19.51, DF=5, p-value = 0.00					
Multivariate Binary Logistic Regression Results for CK Metrics Suite, Model #2						
	Constant	**ck cbo**	**ck dit**	**ck lcom 98**	**ck noc**	**ckwmc-mccabe**
Coeff	-2.98	0.06	0.61	-0.12	0.47	0.02
SE Coeff.	0.61	0.13	0.41	0.20	0.46	0.01
odds ratio	-	1.06	1.85	0.89	1.60	1.02
p-value	0.00	0.64	0.14	0.54	0.31	0.00
LL	-29.26, Tests that all slopes are zero: G=24.34, DF=5, p-value=0.00					
Multivariate Binary Logistic Regression Results for CK Metrics Suite, Model #3						
	Constant	**ck cbo**	**ck dit**	**ck lcom 98**	**ck noc**	**ck rfc**
Coeff	-3.11	0.03	0.71	-0.27	0.54	0.08
SE Coeff.	0.62	0.13	0.42	0.20	0.43	0.02
odds ratio	-	1.03	2.03	0.76	1.72	1.08
p-value	0.00	0.80	0.09	0.17	0.21	0.00
LL	-28.97, Tests that all slopes are zero: G=24.93, DF=5, p-value=0.00					

TABLE 12
Rhino Version 14R3 MOOD Multivariate Binary Logistic Regression Results

Multivariate Binary Logistic Regression Results for MOOD Suite Model #1				
	Constant	**mood ahf**	**mood aif**	**mood mhf**
Coeff	-1.42	1.157	1.285	-1.164
SE Coeff.	0.53	1.206	7.731	0.713
odds ratio	-	3.18	3.62	0.31
p-value	0.008	0.337	0.079	0.102
LL	-38.744, Test that all slopes are zero: G = 5.382, DF = 3, P-Value = 0.146			
Multivariate Binary Logistic Regression Results for MOOD Suite Model #2				
	Constant	**mood ahf**	**mood mhf**	**mood mif**
Coeff	-1.40	1.51	-1.2460	1.5625
SE Coeff.	0.53	1.20	0.7303	0.7926
odds ratio	-	3.16	0.29	4.77
p-value	0.008	-.34	0.088	0.049
LL	-38.354, Test that all slopes are zero: G = 6.163, DF = 3, P-Value = 0.104			

- CK model 1: CBO, DIT, LCOM98, NOC, and WMC.
- CK model 2: CBO, DIT, LCOM98, NOC, and WMC-McCabe.
- CK model 3: CBO, DIT, LCOM98, NOC, and RFC.

A similar analysis was performed for the MOOD and QMOOD metrics. Due to space considerations, their details are not shown, but the resulting binary logistics regression analysis model configurations to be explored are:

- MOOD model 1: AHF, AIF, MHF.
- MOOD model 2: AHF, MHF, MIF.
- QMOOD model 1: CIS, DAM, DCC, MFA.
- QMOOD model 2: DAM, DCC, MFA, NOM.

Multivariate binary logistic regression results. Three models were constructed using the CK metrics. The UBLR analysis over six versions of Rhino showed that CK-RFC, CK-WMC, and CK-WMC-McCabe were consistently good indicators of quality, being statistically significant (at $\alpha = 0.05$) in all cases.

The results of the first model (refer to Table 11) using CK-CBO, CK-DIT, CK-LCOM98, CK-NOC, and CK-WMC show that the log likelihood test statistic G indicates there is sufficient evidence that at least one of the coefficients is different from zero since its p-value is below our critical value of 0.025. However, we note that CK-WMC is the only significant regressor in the model. Since the UBLR for the model with CK-WMC is also significant, then we should ask whether we should just use the UBLR CK-WMC model. If the objective is to use a model with the least number of variables, then we remove all variables from the model that are not significant. However, we intend to use multivariate models throughout all the versions of the Rhino software. Since the level of significance of some of the regressors varies from version to Rhino version and some are significant under the UBLR (e.g., CK-CBO is significant for versions 14R3, 15R1, 15R3, 15R4, and 15R5; CK-LCOM98 is significant in versions 15R2, 15R3, 15R4, and 15R5; CK-DIT is significant in version 15R1, and 15R4; CK-NOC is significant in version 15R3 and 15R5), we conclude that it would be beneficial to include them all into models and carry our analysis forward to all versions of Rhino. Our analysis of models 1 and 2 of the MOOD class metrics suite was not as successful as the CK models. The test statistic G indicates that there is evidence that all of the regressors in either model are not different from zero (refer to Table 12). Also, only MOOD-MIF was significant in model 2. Despite these observations, for the same reasons given in the

TABLE 13
RHINO Version 14R3 QMOOD Multivariate Binary Logistic Regression Results

Multivariate Binary Logistic Regression Results for QMOOD Model #1					
	Constant	**qmood cis**	**qmood dam**	**qmood dcc**	**qmood mfa**
Coeff	-3.08	0.055	0.63	-0.08	1.18
SE Coeff.	1.72	0.023	1.70	0.07	0.81
odds ratio	-	1.06	1.87	0.93	3.26
p-value	0.07	0.02	0.71	0.30	0.14
LL	-36.650, Test that all slopes are zero: G = 9.571, DF = 4, P-Value = 0.048				
Multivariate Binary Logistic Regression Results for QMOOD Model #2					
	Constant	**qmood dam**	**qmood dcc**	**qmood mfa**	qmood nom
Coeff	-4.34	1.394	-0.11	1.32	**0.08**
SE Coeff.	2.11	2.06	0.08	0.87	**0.02**
odds ratio	-	4.03	0.90	3.73	**1.08**
p-value	0.04	0.50	0.14	0.13	**0.00**
LL	-32.283, Test that all slopes are zero: G = 18.305, DF = 4, P-Value = 0.001				

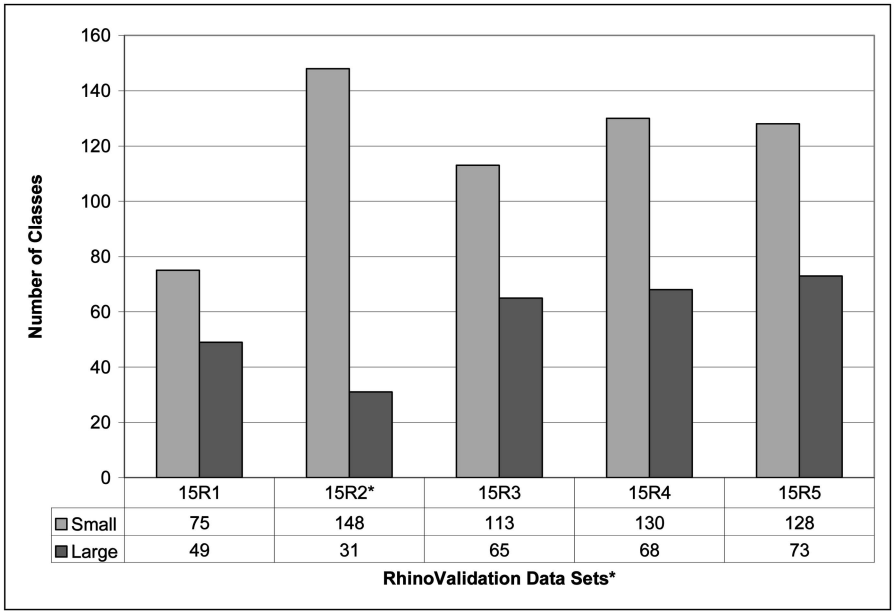


Fig. 3. Rhino validation set sizes.

CK models, we carry both of these models forward in our validation analysis.

Finally, we analyze the MBLR results of the QMOOD metrics (refer to Table 13). The G statistics for both models show that there is sufficient evidence that at least one of the coefficients in each model is different from zero since its p-value is below our critical value of 0.025. In addition, there is one regressor in each model that is significant. Therefore, for the same reasons given for the CK metrics, we carry both models forward in our validation analysis.

Model validation of MBLR models. We used a simple holdout method to validate our CK, MOOD, and QMOOD MBLR models. We developed MBLR models with Rhino version n data and validated them with Rhino version $n + 1$ data for versions 14R3 through 15R5.

The measure of effectiveness used is the concordant, discordant, and tied pairs. The concordant, discordant, and tied pairs entries are calculated by pairing the observations with different response values: the pairs, concordant, discordant, and tied pairs. For example, Rhino version 14R3 has 15 classes with defects and 80 classes that are not defective. This results in $80 * 15$ pairs with different response values. A pair is concordant if a class that is defective has a higher probability of being defective, discordant if the opposite is true, and tied if the probabilities are equal. In Rhino version 14R3 using CK-WMC, 83.6 percent of the pairs are concordant, 15.8 percent are discordant, and 0.07 percent are tied. These values will be used to validate the model with data from other versions of Rhino.

In order to control for class size, we examined the performance of the models using all the classes, small classes (simple $\mathrm{WMC} \leq 10$), and then large classes (simple $\mathrm{WMC} > 10$). Fig. 3 and Fig. 4 show the sizes and errors, respectively, in the Rhino validation sets used to control for size. The asterisk $(*)$ in Rhino version 15R2 in both figures indicates that the $\mathrm{WMC} \leq 10$ threshold was exceeded for the small data set since all of the small classes $(\mathrm{WMC} \leq 10)$ were fault-free. In this case, just enough classes, in ascending order of WMC, were added until one class, which contained a fault, was included in the small data set. This allowed the statistical software to function properly over the small data set.

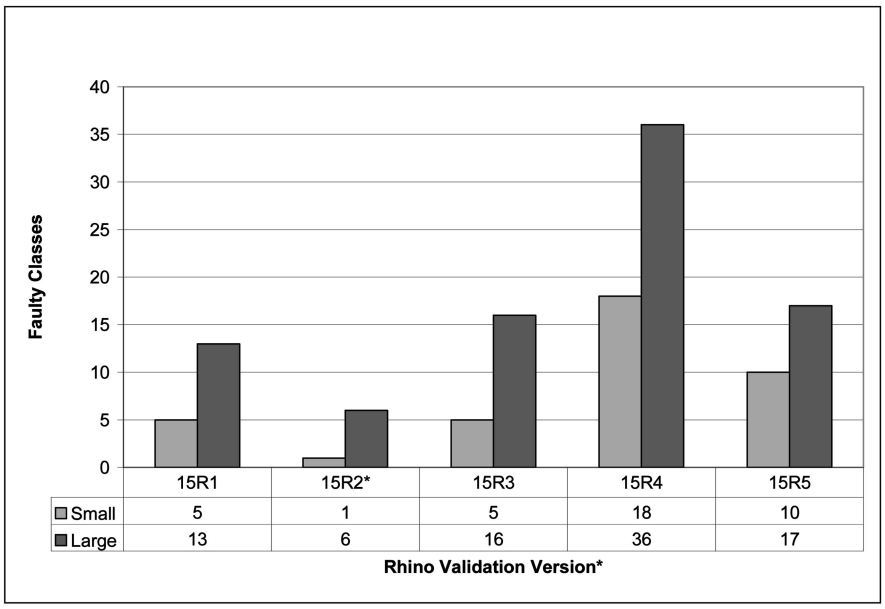


Fig. 4. Rhino errors per validation data set.

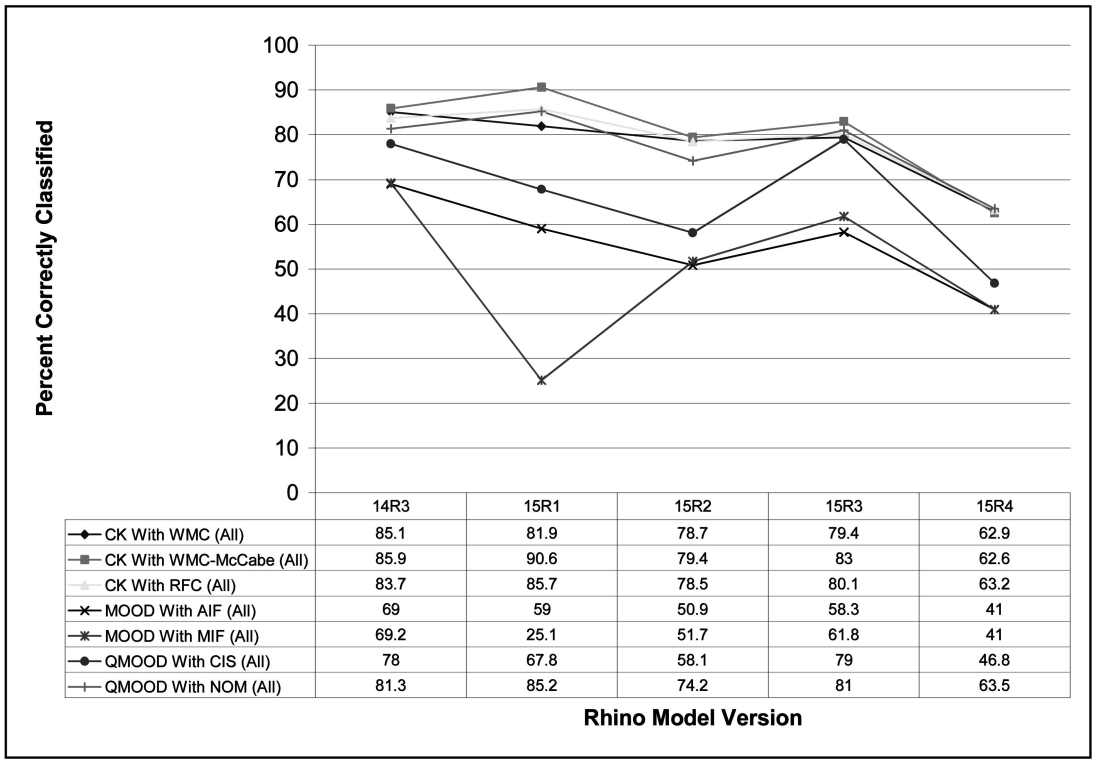
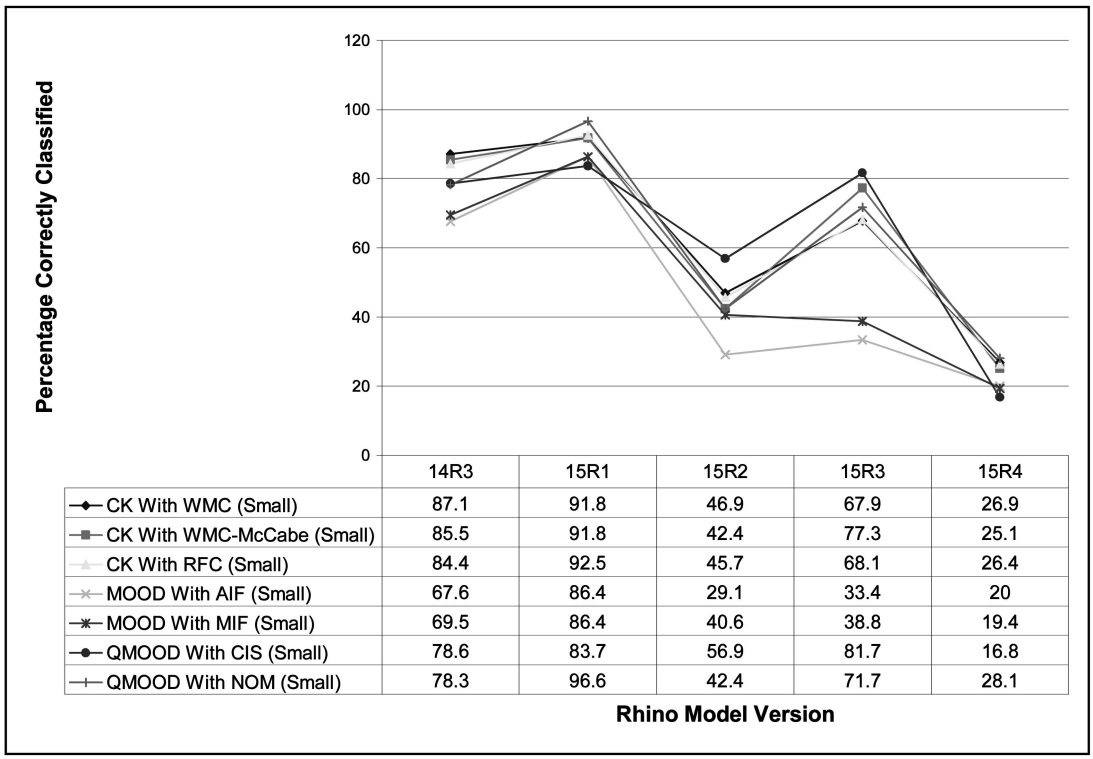


Fig. 5. Rhino model validation—all models.

Therefore, we consider the Rhino 15R2 small validation data set to be biased and will not rely heavily on its results. We will rely more on the other data points.

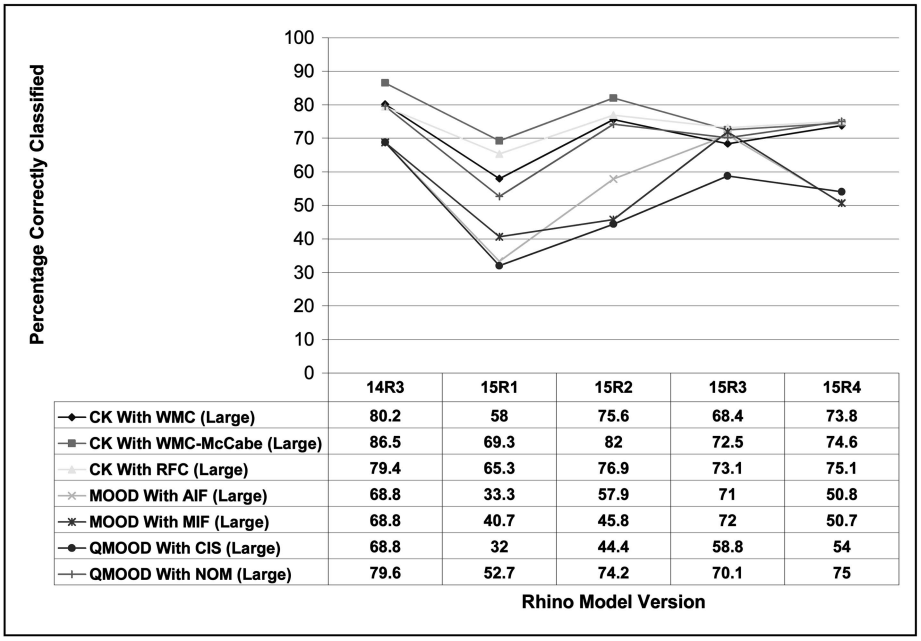
Figs. 5, 6, 7, 8, 9, 10 display only the concordant values in graphical form. Fig. 5 shows the overall performance of the models. Based on the results obtained for the validation of Rhino models 14R3, we conclude that there is evidence to support Hypothesis 1, that OO software metrics can identify fault-prone classes in traditional and highly iterative, or agile, software projects during its initial delivery (initial quality). The performance of all the metrics suites models was between 69.0 percent and 85.9 percent accuracy. There is also evidence to support Hypothesis 2, that OO software metrics can identify fault-prone classes in multiple, sequential versions of highly iterative, or agile, software projects. All of our CK metrics suites models, and one QMOOD metrics suites model, were able to correctly classify fault-prone classes in all versions of the Rhino software. The accuracy range of these predictive models was between 62.6 percent and 90.6 percent.

Our secondary findings are that there is a general deterioration of the effectiveness of the metrics as the Rhino software progresses from 14R3 through 15R4 (five validation versions). The performance of the MOOD metrics



	14R3	15R1	15R2	15R3	15R4
CK With WMC (Small)	87.1	91.8	46.9	67.9	26.9
CK With WMC-McCabe (Small)	85.5	91.8	42.4	77.3	25.1
CK With RFC (Small)	84.4	92.5	45.7	68.1	26.4
MOOD With AIF (Small)	67.6	86.4	29.1	33.4	20
MOOD With MIF (Small)	69.5	86.4	40.6	38.8	19.4
QMOOD With CIS (Small)	78.6	83.7	56.9	81.7	16.8
QMOOD With NOM (Small)	78.3	96.6	42.4	71.7	28.1

Fig. 6. Rhino model validation performance of small models $(\mathrm{WMC} \leq 10)$.



	14R3	15R1	15R2	15R3	15R4
CK With WMC (Large)	80.2	58	75.6	68.4	73.8
CK With WMC-McCabe (Large)	86.5	69.3	82	72.5	74.6
CK With RFC (Large)	79.4	65.3	76.9	73.1	75.1
MOOD With AIF (Large)	68.8	33.3	57.9	71	50.8
MOOD With MIF (Large)	68.8	40.7	45.8	72	50.7
QMOOD With CIS (Large)	68.8	32	44.4	58.8	54
QMOOD With NOM (Large)	79.6	52.7	74.2	70.1	75

Fig. 7. Rhino model validation performance of large models $(\mathrm{WMC} > 10)$.

models was least effective while the CK metrics models performed best overall using all of the validation data (not dividing the data by size). The QMOOD metrics suite model 2 (using QMOOD-NOM) was also as good as the CK metrics models. The QMOOD metric suite model (using QMOOD-CIS) did not perform as well but was better than both MOOD metrics models.

Our experiment also took class size into consideration. Analyzing all of the models using small classes (simple $\mathrm{WMC} \leq 10$) showed their effectiveness deteroriated much faster through successive versions of Rhino than the models using all of the validation data (refer to Fig. 6). Analyzing all of the models using large classes (simple $\mathrm{WMC} > 10$) shows that the models are significantly more effective over the multiple sequential versions of Rhino than when they are used to assess small classes (refer to Fig. 7).

Figs. 8, 9, and 10 show individual model validation performance for small, large, and all classes combined. Fig. 8 shows the CK metrics suite model 1, Fig. 9 shows the MOOD metrics suite model 2, and Fig. 10 shows the QMOOD metrics suite model 1. We observe that, in the early versions of Rhino, the CK and QMOOD metrics suite

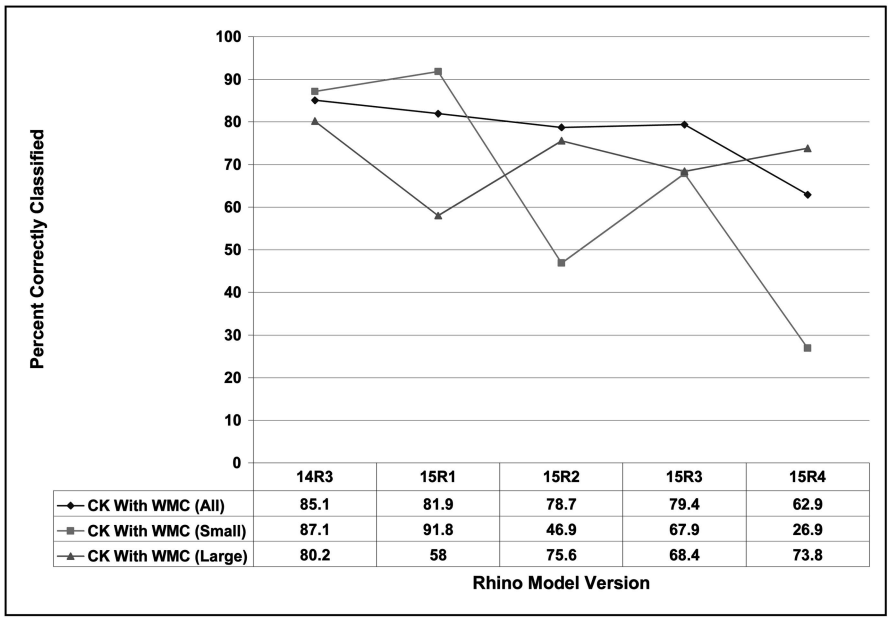


Fig. 8. Rhino CK-WMC model validation performance (small and large).

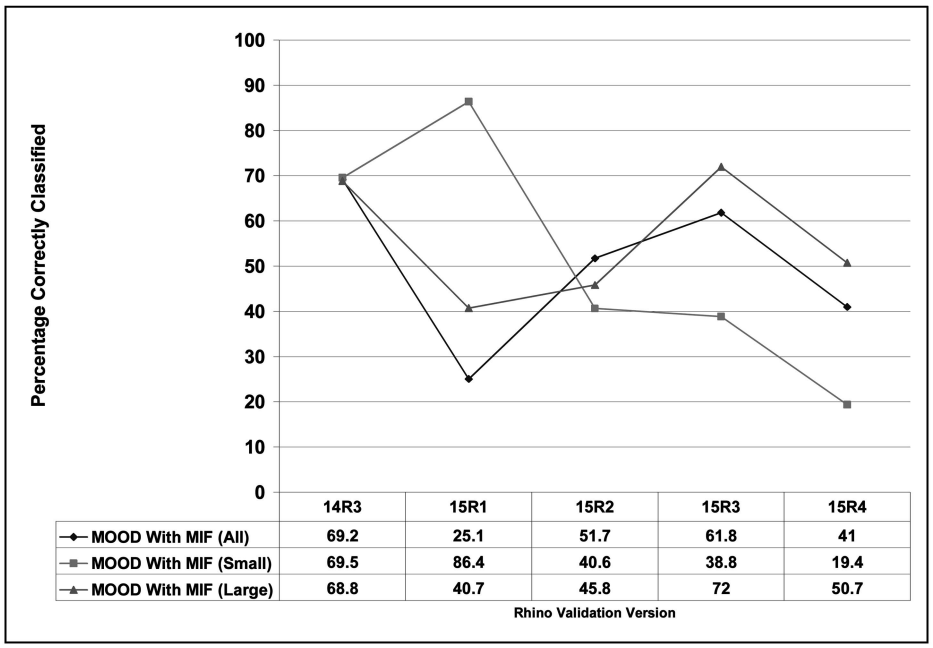


Fig. 9. Rhino MOOD-MIF model validation performance (small and large).

models performed best over small classes. However, in the later versions of Rhino, the metrics suites performed best over large classes.

4.4 Discussion

The CK and QMOOD metrics suites models performed well overall, while the performance of the MOOD metrics suite models trailed significantly. In general, the performance of the metrics suites in an agile software development context performed well and there was evidence to support both our research hypotheses. To support Hypothesis 2, there seems to be a significant amount of change in terms of new and changed classes in successive Rhino versions. Fig. 11 depicts this change graphically. Class changes play a significant role in the development of the Rhino software from one version of Rhino to the next. When there are class changes and when new classes are introduced there is a greater chance of introducing errors than when software is neither being changed nor enhanced. So, the dynamic nature of the software process enables the metrics suites to be effective in predicting fault-prone classes.

The issue of the metrics suites' effectiveness on small classes versus large classes is a matter that requires more careful analysis. In analyzing the performance of the models over small and large class sizes, we refer to Fig. 4 and note that an increase of faulty classes in both the small and large class validation data sets produced increases in the performance of the models. However, the performance on these varies significantly. If we examine the performance of CK model 1 over the small class validation data sets,

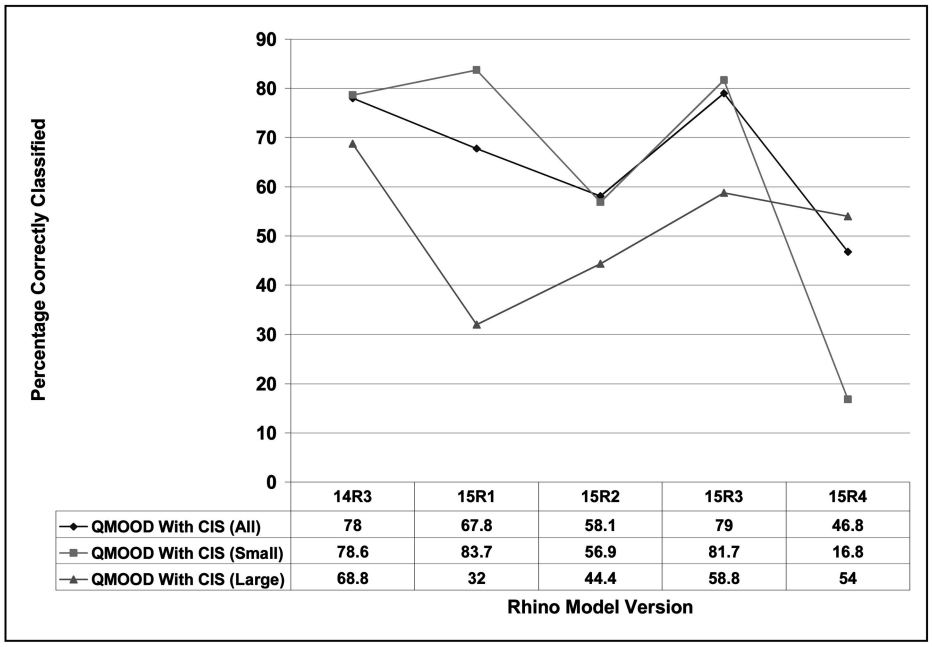


Fig. 10. Rhino QMOOD-CIS model validation performance (small and large).

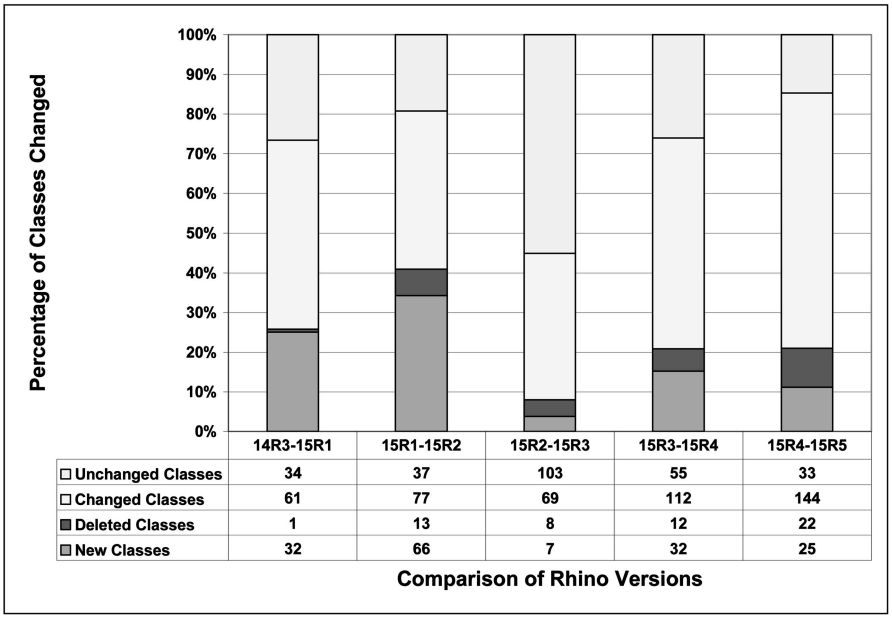


Fig. 11. Rhino comparison of versions showing unchanged, changed, deleted and new classes.

specifically in versions 15R1 and 15R3, we note that there are five errors in these validation versions, but the performance results are different (87.1 percent versus 46.9 percent, respectively). If we compare this result to that of CK model 1 over large classes, we note that there are 13 errors in validation data set 15R1 and 16 errors in validation data set 15R3. The model performance decreases slightly from 80.2 percent to 75.6 percent, respectively. The effectiveness decrease in the metrics is not surprising as the Rhino software is continuously improving in quality and errors are being removed from every version. However, what is surprising is the hypersensitivity of the performance of the CK model 1 over small classes as compared to its performance over large classes. Recall that we are using the same model and the same validation set versions. Although not shown due to space limitations, CK models 2 and 3 and QMOOD model 2 follow the same pattern. Another observation, is that when we try the same analysis on the full data sets for these validation versions of Rhino, we see a pattern that tends to follow that of the large class validation data set. However, there are significantly larger numbers of small classes in both the 15R1 and 15R3 validation data sets. We conclude that the performance effectiveness of the metric suite models deteriorates over the multiple, sequential versions of Rhino and that they are hypersensitive (are not steady) when applied to small classes. That is, small changes in the number of faulty classes cause big changes in model performance. Their performance is steadier when applied to large classes or a mixture of large and small classes since changes in the number of faulty classes cause a proportional small change in model performance.

Both MOOD metrics suites models and QMOOD metrics suite model 1 (refer to Fig. 10) do not follow the pattern of the CK metric suites models. Both MOOD metrics suite models performed best on small classes in the first two versions of Rhino, and then deteriorated badly. The performance of both MOOD metrics suite models over large models was identical and showed a greater sensitivity to class size than did the CK metrics suite models. The MOOD metrics suites models can be said to be a direct measure of *size* when used over large classes.

5 Conclusions

We conducted a statistical analysis of the CK, MOOD, and QMOOD OO class metrics suites using six versions of Mozilla's Rhino open source software. The results of this case study indicate the CK and QMOOD OO class metrics suites are useful in developing quality classification models to predict defects in both traditional and highly iterative, or agile, software development processes for both initial delivery and for multiple, sequential releases. This is the primary contribution of our research since few studies using OO metrics to predict defects in highly iterative, or agile, software have been performed previously. Another contribution of our paper is an empirical study of the QMOOD and MOOD metrics since few empirical studies of these metrics have been performed in the past.

Our UBLR analysis indicated that the individual metrics CK-WMC, CK-RFC, QMOOD-CIS, and QMOOD-NOM are consistent predictors of class quality (error-proneness). The UBLR analysis for the MOOD metrics indicated that the MOOD metrics, in general, were not useful as predictors of OO class quality (error-proneness).

Our MBLR analysis indicated that the CK metrics suite produced the best three models for predicting OO class quality (error-proneness), followed closely by one QMOOD model. We did not mix individual class metrics from the CK and QMOOD metrics suites because 1) users would often have one of these metrics suites, not both, and 2) because many of the CK and QMOOD class metrics are highly correlated, indicating that they measure the same dimension of the software classes.

One implication of our results for software practitioners is that the CK metrics have been shown to be better and more reliable predictors of fault-proneness than the MOOD or QMOOD metrics. Also, class size can impact metric performance. Thus, when the software is in the early stages of evolution and complexity is still low, the metrics will not be very effective. Also, for highly iterative or agile systems, the metrics will not remain effective forever. There are practical limitations to the effectiveness of the metrics over the course of several software iterations as the software matures and the dynamic nature of the software development process subsides.

6 Future Research

In the future, we plan to investigate two areas:

- whether complexity-related measures are effective in detecting error-prone classes in highly iterative or agile processes, and
- whether decision trees are more effective than binary logistic regression in detecting error-prone classes in highly iterative or agile processes.

Most of the metrics that correlated highly with error-proneness in this study had some association with complexity. CK-WMC and CK-WMC-McCabe are complexity metrics. CK-RFC is, in an indirect sense, also a complexity metric and has a very large correlation with CK-WMC. QMOOD-CIS is also closely related to CK-WMC as it is a count of the public methods in a class and is therefore a complexity metric. QMOOD-NOM is identical to CK-WMC. It would be of interest to determine if various aspects of OO complexity proposed in previous studies and implemented in various metrics suites are better predictors of OO class quality in highly iterative or agile systems. In addition, it would also be of interest to investigate the use of decision trees using metrics from this study. Decision trees are more flexible and robust than statistical classification models and can be developed with fewer restrictions.

Acknowledgments

The authors thank Norris Boyd of the Mozilla Rhino Project for providing invaluable support and information about the Rhino software, and Stacy Lukins of the University of Alabama in Huntsville for her invaluable assistance analyzing the Rhino software and collecting and organizing the Rhino class fault data. Finally, we thank the anonymous reviewers of this paper for their thoughtful and generous comments. This work was funded in part by the US National Aeronautics and Space Administration under Grants NAG5-12725 and NCC8-200.

References

[1] J. Bansiya and C. Davis, "A Hierarchical Model for Object-Oriented Design Quality Assessment," *IEEE Trans. Software Eng.*, vol. 28, no. 1, pp. 4-17, Jan. 2002.

[2] F. Brito e Abreu and W. Melo, "Evaluating the Impact of Object-Oriented Design on Software Quality," *Proc. Third Int'l Software Metrics Symp.*, pp. 90-99, 1996.

[3] S. Chidamber and C. Kemerer, "A Metrics Suite for Object-Oriented Design," *IEEE Trans. Software Eng.*, vol. 20, no. 6, pp. 476-493, June 1994.

[4] L.H. Etzkorn, S. Gholston, J.L. Fortune, C.E. Stein, D. Utley, P.A. Farrington, and G.W. Cox, "A Comparison of Cohesion Metrics for Object-Oriented Systems," *Information and Software Technology*, vol. 46, no. 10, pp. 677-687, Aug. 2004.

[5] P. Nesi and T. Querci, "Effort Estimation and Prediction of Object-Oriented Systems," *J. Systems and Software*, vol. 42, pp. 89-102, July 1998.

[6] V. Basili, L. Briand, and W. Melo, "A Validation of Object-Oriented Design Metrics as Quality Indicators," *IEEE Trans. Software Eng.*, vol. 22, no. 10, pp. 751-761, Oct. 1996.

[7] K. El Emam, S. Benlarbi, N. Goel, and S. Rai, "The Confounding Effect of Class Size on the Validity of Object-Oriented Metrics," *IEEE Trans. Software Eng.*, vol. 27, no. 7, pp. 630-650, July 2001.

[8] L. Briand, J. Daly, and J. Wuest, "A Unified Framework for Coupling Measurement in Object-Oriented Systems," *IEEE Trans. Software Eng.*, vol. 25, no. 1, pp. 91-121, Jan.-Feb. 1999.

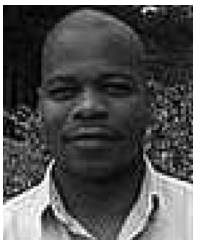
[9] A. Binkley and S. Schach, "Validation of the Coupling Dependency Metric as a Predictor of Run-Time Failures and Maintenance Measures," *Proc. 20th Int'l Conf. Software Eng.*, pp. 452-455, 1998.

[10] A. Jensen, *An Overview and Evaluation of Object-Oriented Software Metrics Used in Cantata++ Static Analysis.* Information Processing Ltd., http://www.qcsltd.com/cantpp/cantpp.htm, Apr. 2004.

[11] R. Subramanyam and M.S. Krishnan, "Empirical Analysis of CK Metrics for Object-Oriented Design Complexity: Implications for Software Defects," *IEEE Trans. Software Eng.,* vol. 29, no. 4, pp. 297-310, Apr. 2003.
[12] P. Bellini, I. Bruno, P. Nesi, and D. Rogai, "Comparing Fault-Proneness Estimation Models," *Proc. 10th IEEE Int'l Conf. Eng. Complex Computer Systems,* pp. 205-214, 2005.
[13] L. Briand, W. Melo, and J. Wuest, "Assessing the Applicability of Fault-Proneness Models across Object-Oriented Software Projects," *IEEE Trans. Software Eng.,* vol. 28, no. 7, pp. 706-720, July 2002.
[14] G. Denaro and M. Pezze, "An Empirical Evaluation of Fault-Proneness Models," *Proc. 24th Int'l Conf. Software Eng.,* pp. 241-251, 2002.
[15] R. Ferenc, I. Siket, and T. Gyimothy, "Empirical Validation of Object-Oriented Metrics on Open Source Software for Fault Prediction," *IEEE Trans. Software Eng.,* vol. 31, no. 10, pp. 897-910, Oct. 2005.
[16] N. Boyd, Rhino Home Page, http://www.mozilla. org/rhino/, July 2006.
[17] N. Boyd, Questions about Rhino History and Software Development Process, e-mail to the author, 6 Sept. 2005.
[18] M. Cohn, Agile Alliance Home Page, http://www.agilealliance. org, Apr. 2006.
[19] F. Brito e Abreu, G. Pereira, and P. Soursa, "Coupling-Guided Cluster Analysis Approach to Reengineer the Modularity of Object-Oriented Systems," *Proc. Euromicro Conf. Software Maintenance and Reeng.,* pp. 13-22, 2000.
[20] S. Quattlebaum, "A Comparison of the Results of Object-Oriented Metrics in C++ and Java," MS thesis, Univ. of Alabama in Huntsville, 2004.
[21] Information Processing Ltd., *Cantata++ User's Manual.* Information Processing Ltd., http://www.iplbath.com/products/p0000. shtml, Apr. 2004.
[22] R. Harrison, S. Counsell, and R. Nithi, "An Evaluation of the MOOD Set of Object Oriented Software Metrics," *IEEE Trans. Software Eng.,* vol. 24, no. 6, pp. 150-157, June 1998.
[23] Bugzilla Database, Mozilla Foundation, https://bugzilla.mozilla. org/, July 2004.
[24] D. Hosmer and S. Lemeshow, *Applied Logistic Regression,* second ed. John Wiley and Sons, 2000.
[25] W.G. Hopkins, *A New View of Statistics.* Sport Science, http:// www.sportsci.org/resource/stats, Apr. 2003.
[26] Information Systems/Technical Academic Support, Center for Statistical Computing Support, Social Sciences Teaching and Research Statistics (STARS), Univ. of Kentucky, http://www.uky. edu/ComputingCenter/SSTARS/MulticollinearityinLogistic Regression.htm, Oct. 2006.
[27] XLMiner Online Help,Cytel Statistical Software, Resampling Stats, http://resample.com/xlminer/help/Lreg/lreg_ex.htm, Oct. 2006.

Hector M. Olague received the bachelor's degree in marine engineering systems from the US Merchant Marine Academy and the MS and PhD degrees in computer science from the University of Alabama in Huntsville. He is an engineer and a government program manager at the US Army Space and Missile Defense Command in Huntsville, Alabama. His research interests include software engineering, object-oriented software metrics, information theory, and statistical and nonstatistical classification modeling.

Letha H. Etzkorn received the bachelor's and master's degree in electrical engineering from the Georgia Institute of Technology and the PhD degree in computer science from the University of Alabama in Huntsville. She is an associate professor in the Computer Science Department at the University of Alabama in Huntsville. Her primary research areas are in software engineering, primarily software metrics and program understanding, and mobile and intelligent agents. She is a senior member of the IEEE.



Sampson Gholston received the PhD in industrial and systems engineering from the University of Alabama in Huntsville and the master's from the University of Alabama in industrial engineering. He is an associate professor at the University of Alabama in Huntsville. He has also worked for Saturn Corporation as a supplier quality engineer. His research interests include quality engineering, quality management, and applied statistics.

Stephen Quattlebaum received the BS degree in computer science from the University of Alabama and the MS degree in computer science from the University of Alabama in Huntsville. He is senior enterprise developer at DragonFly Athletics LLC. His primary research interests include software engineering and design of programming languages and environments.