CHAPTER 5

METRIC SUITES VALIDATION USING FAULT PRONENESS

5.1 INTRODUCTION

Object-oriented design has emerged as a dominant method in software industry and many new metrics have been proposed for quality prediction of object-oriented programs, but the significance of those metrics is not yet confirmed. Software process control can be improved and high software reliability can be achieved if faults are predicted early in the software life cycle. Testing quality related issues of software has become critical with the increasing importance of the quality of software. Many authors have suggested theoretical validation followed by empirical evaluation(Briand 2001) using proven statistical and experimental techniques for evaluating in the field of usefulness and relevance of any new metrics.

In the proposed research work an empirical validation of software quality metric suites on open source software for fault-proneness prediction in object oriented system has been presented. The three metrics used here are Chidamber and Kemerer metrics, Robert C. Martin metric suite and McCabe's metric suite. Using these metrics suite, the defects present in different versions of Rhino software have been analyzed to predict the software quality by making use of the fault proneness concept. From the results and empirical analysis, it is clear that the different metric suites have different efficiency in faults prediction. With the aid of this empirical

analysis, it is suggested that the software professionals need to find out those metric suites that can predict faults while developing the metric-quality software products using the OO approach.

5.2 QUALITY METRIC SUITES VALIDATED IN THE RESEARCH

The three different quality metric suites are empirically validated using the software Rhino. The detailed description of the metrics in the validated metric suites is given in the following sub-sections. The block diagram of the metric suites validated in the proposed approach is depicted in Figure 5.1.

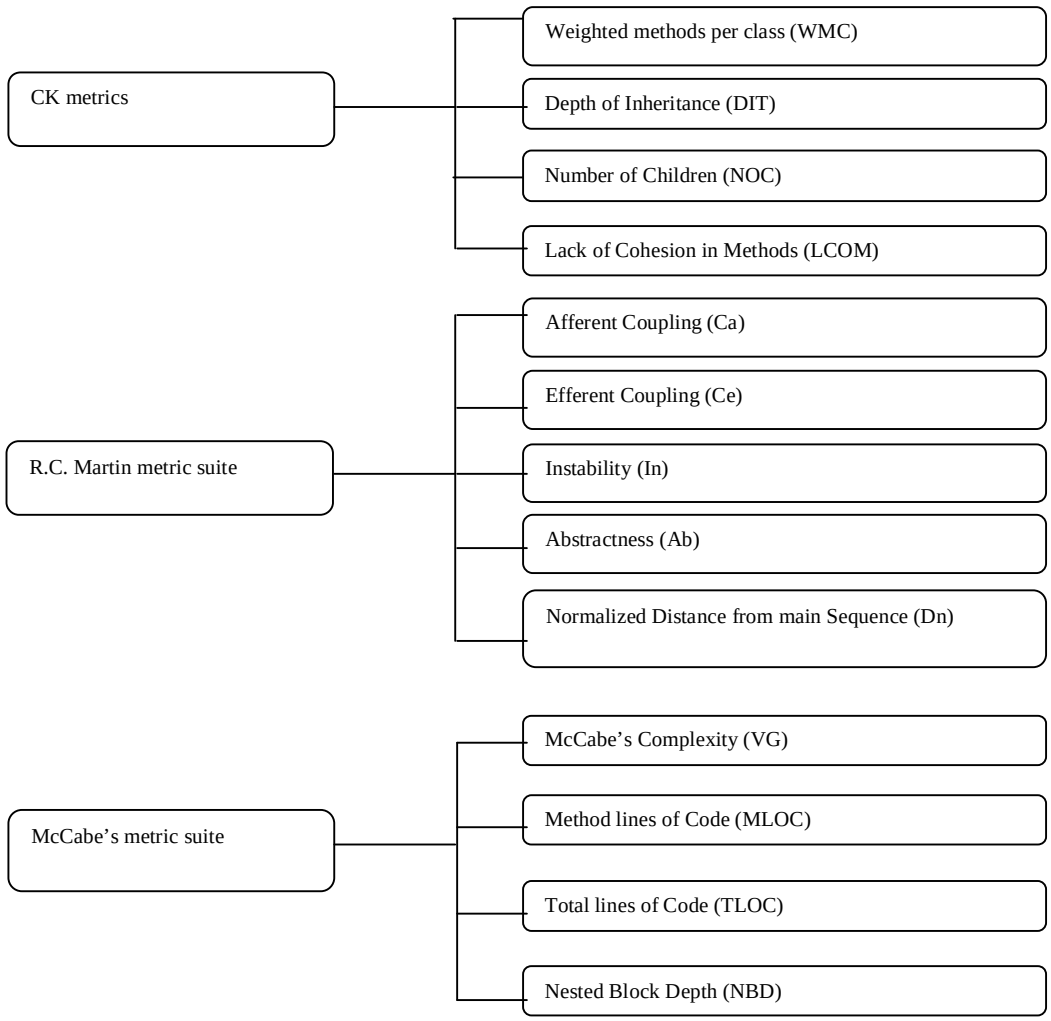

Figure 5.1 Metric suites validated in the proposed approach

5.2.1 Chidamber and Kemerer Metrics

"Classic" set of metrics proposed by Chidamber and Kemerer in 1991 (Chidamber 1991) particularly for object-oriented software has been upgraded in 1994 Chidamber et al (1994). The creators of this group of metrics say that these measures can assist users in understanding object oriented design complexity and in forecasting external software qualities for example software defects, testing, and maintenance effort (Korson 1990). Utilization of the CK set of metrics and other corresponding measures are progressively growing in industry for acceptance. The Chidamber and Kemerer metrics suite is an extensively used metrics suite. It has been certified by a number of researchers and it forms the heart of the study.

*** Weighted Methods per Class**

The WMC of a class is the weighted sum of the complexities of its methods. More accurately, WMC is defined to be the weighted number of all member functions and operators defined in each class Chidamber et al (1994).

*** Depth of Inheritance Tree**

The depth of a class within the inheritance hierarchy is the highest number of steps from the class node to the root node of the tree and it measures the number of ancestor classes. The number of methods inherited by a class increases with the depth of the class within the hierarchy, making it more difficult to forecast its behavior Chidamber et al (1994).

*** Number of Children**

According to this metric, Number of Children of a class is the number of immediate sub-classes subordinated to a class in the class hierarchy. NOC metric is theoretically based on the idea related to the scope of properties. It measures the number of sub-classes that are going to inherit the methods of the parent class. As NOC metric counts only the immediate sub-classes instead of all its descendants, the definition of NOC metric presents a changed view of the system Chidamber et al (1994).

*** Coupling between Objects**

CBO is a measure of the number of other classes to which a class is coupled. Two classes are said to be coupled if methods and variables defined in one class are used by methods declared in the other class. In depth information about the methods, in other words detailed design description or the class code is required by this metric for making measurement Chidamber et al (1994).

(v) Response for a Class

The response set of a class is composed of the set of all the methods of the class and the set of methods called directly by the methods (i.e., the set of methods that can potentially be executed in response to a message received by an object of that class). This includes all methods accessible within the class hierarchy. This metric utilizes the number of methods to evaluate the complexity of the class and also the amount of communication it has with other classes Chidamber et al (1994).

*** Lack of Cohesion in Methods**

Lack of cohesion measures how well the methods in a class are unrelated to each other(Hitz 1995). Methods in a cohesive class use the same set of variables. A highly cohesive module should be independent; high cohesion signifies good class subdivision. Lack of cohesion or low cohesion adds to complexity, thereby increasing the chances of errors during the development process. High cohesion signifies simplicity and high reusability. Classes with low cohesion could possibly be reconstructed into two or more smaller classes with increased cohesion Chidamber et al (1994).

5.2.2 Robert C. Martin's Metric Suite

*** Efferent Coupling**

Efferent coupling between packages measures the total number of external classes coupled to classes of a package because of outgoing coupling (external classes used by package's class). All classes are counted only once. If the package does not contain any class or the package's classes do not use any external class, then the value of Ce is zero. Ce is mainly appropriate for object-oriented systems (Robert 2004).

*** Afferent coupling**

Afferent coupling between packages measures the total number of external classes coupled to classes of a package because of incoming coupling (Briand 1999). All classes are counted only once. If the package does not contain any class or external classes do not use any of the package's classes, then the value of Ca is zero. Ca is mainly appropriate for object-oriented systems.

*** Instability**

Instability is the ratio of efferent coupling to total coupling (Ce + Ca) such that In = Ce / (Ce + Ca). This metric signifies the package's adaptability to change. The range for this metric is 0 to 1, with I=0 signifying an absolutely stable package and I=1 signifying an absolutely instable package (Robert 2004).

*** Abstractness**

Abstractness (Ab) is the ratio of the number of abstract classes (and interfaces) to the total number of classes in the evaluated package. The range for this metric is 0 to 1, with Ab=0 signifying an absolutely concrete package and Ab=1 signifying an absolutely abstract package.

*** Normalized Distance from Main Sequence**

Normalized distance cdp from main sequence is the perpendicular distance of a package from the idealized line A + I = 1, where A is a measure of abstractness and I is a measure of instability. Ideal packages are either absolutely abstract or stable (x=0, y=1) or absolutely concrete and unstable (x=1, y=0). The range for this metric is 0 to 1, with D=0 representing a package that is coincident with the main sequence and D=1 representing a package that is as far away from the main sequence as possible (Robert 2004).

5.2.3 McCabe's Metric Suite

The cyclomatic complexity (McCabe 1976) is used to evaluate the complexity of an algorithm in a method. A method with a low cyclomatic complexity need not inevitably mean that the methods are not complex because it may mean that decisions are postponed through message passing.

On account of inheritance, cyclomatic complexity cannot be used to measure the complexity of a class, but cyclomatic complexity of individual methods united with other measures can be used to calculate the complexity of a class.

Though this metric is particularly applicable to the estimation of the complexity feature, it is also associated with all the other features. Number of flows through a section of code is measured by McCabeCC. Whenever a branch takes place, this metric is increased by one, (if, for, while, do, case, catch and the ?: ternary operator, as well as the && and || conditional logic operators in expressions). It is computed for methods only. High value of this metric indicates the complexity of the application or at least the presence of huge number of alternative flows in the application. Cyclomatic complexity is rooted in the program control structure and can be computed from the amount of conditional statements present in the source code (McCabe 1976).

*** Lines of Code**

The LOC of a class is the total number lines in the body of the class and its methods excluding empty and comment lines. LOC metrics calculates total lines of code blank lines of code (BLOC), comment lines of code (CLOC), lines with both code and comments (C&SLOC), logical source lines of code (SLOC-L), McCabe VG complexity (MVG), and number of comment words (CWORDS). Physical executable source lines of code (SLOC-P) are determined by subtracting the number of blank lines and comment lines from the total number of lines of source code. Count for the entire project was computed by adding the individual file count (McCabe 1976). A comment word histogram was also created by the LOC metrics.

*** Method Lines of Code**

A method line of code is the total number of non-blank and non-comment lines present inside a method. Total number of lines of code inside method body includes (average line of code) avg, (Maximum line of code) max, and (Sum of line of code) sum but excludes blank and comment lines.

*** Source lines of code**

SLOC is normally used to forecast the amount of effort that will be necessary to build up a program, as well as to assess programming productivity or effort once the software is created. There are two main types in SLOC, physical and logical. Physical SLOC is the number of lines in the source code of the program including comment lines and blank lines. If a section consists of more than 25% blank lines then the blank lines that are in excess of 25% are ignored. Logical SLOC tries to measure the number of statements instead of lines in a program's source code.

*** Nested Block Depth**

Avg, max, and sum are included in the calculation of the depth of nested blocks of code.

5.3 DESIGN OF THE RESEARCH METHODOLOGY

The research design of empirical study is depicted in Figure 5.2, which suggests that metric suites have an influence on the fault proneness. Fault proneness is the dependent variable whereas metric suites are independent variables. This study is conducted to find the existence of relationship between metric suites and fault proneness, and to develop a model to choose an effective metric suite for open source environment. The empirical study was carried out using the bug report collected from the open source software development industry.

Fault-proneness of a software module is the probability that the module contains faults. A correlation exists between the fault-proneness of the software and the measurable attributes of the code (i.e. the static metrics) and of the testing (i.e. the dynamic metrics). Since testing can be done by various methods, it is termed as dynamic metrics (Kamaljit 2009). Early detection of fault-prone software components enables verification by experts to use their time and resources and concentrate on the problem areas of the software system under development.

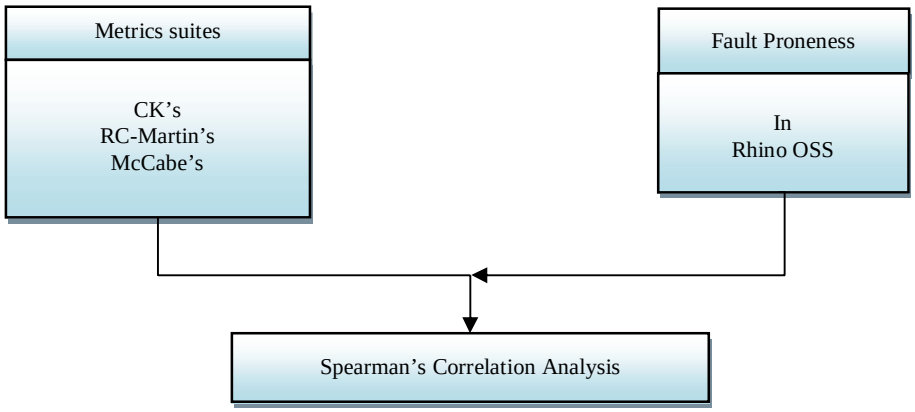

Figure 5.2 Research design of the empirical validation of metric suites

The three metrics used here are Chidamber and Kemerer Metrics, Robert C. Martin Metric Suite and McCabe's Metric Suite. All these metric suites have been individually validated by comparing their values with similar metrics and have been found to be better metrics Amjan et al (2012).

The second independent variable in the study is the fault proneness. Using these metrics suite, the defects present in different versions of Rhino software have been analyzed to predict the software quality by making use of the fault proneness concept.

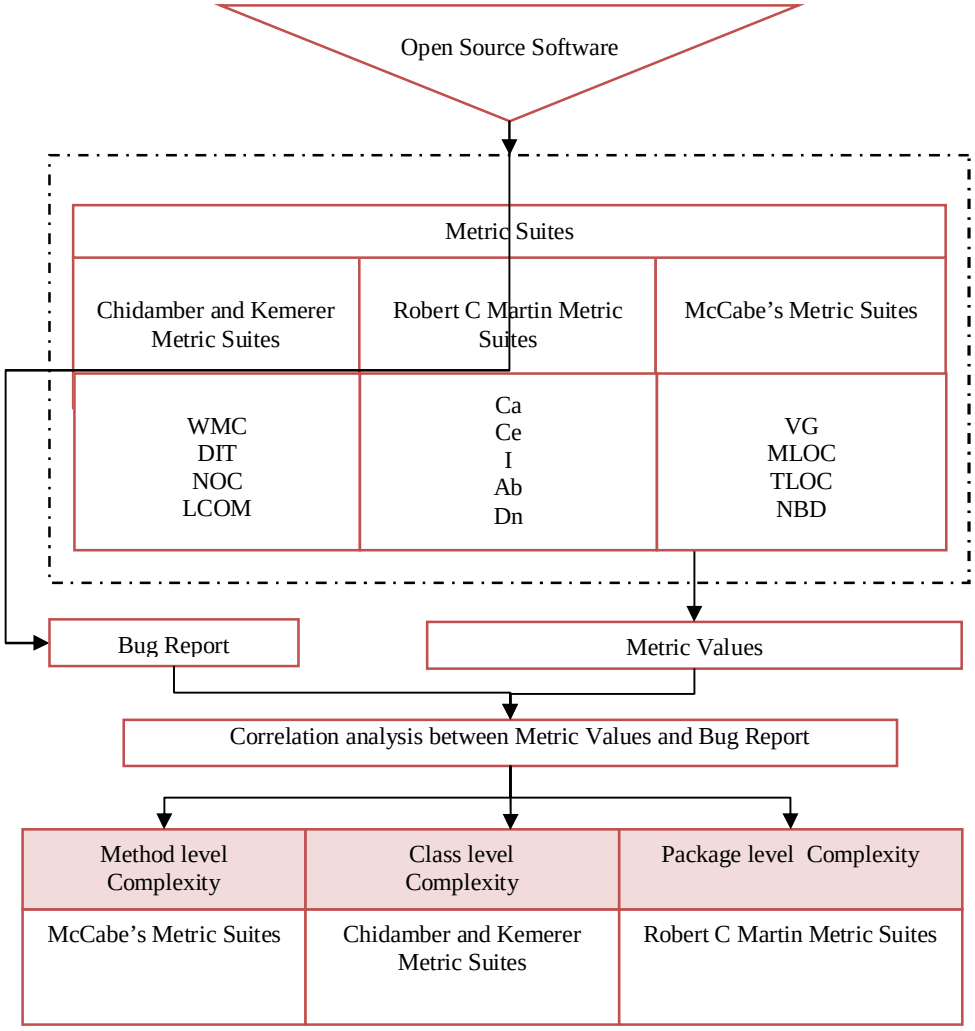

Figure 5.3 Metric Suites and their Relevance

The relevance of the Metric suites with respect to the programming components is measured as shown in Figure 5.3. The source code of the Rhino15R4 is analysed and the metric values are obtained using the CkMetrics.java (Appendix 1). The xml files generated by CkMetrics.java and the bug values obtained are dispensed to a program that supplies the correlated values of each and every metric. Similarly, the correlation values are obtained for the other versions of Rhino (Appendix 2).

The main research objective is to focus on the relationship between the metric suites and the fault proneness as shown in Figure 5.2. The spearman's correlation analysis Jiang et al (2006) is made to find the relation

between the metric suites and the fault proneness of the open source software to identify the suitable metric suite.

There are numerous ways to assess the relationship between two variables. Some of them are t-test/ANOVA, correlation and regression. They can be used to see whether each complexity metric suites is a reliable indicator of fault proneness. The fault proneness from the bug report is evaluated with each of the three metric suite, Chidamber and Kemerer Metrics, Robert C. Martin Metric Suite and McCabe's Metric Suite. Tests are conducted to validate the proposed metric suites empirically (Fredrick 2002). This stands as the primary objective of the experiment.

5.4 RESULTS AND DISCUSSIONS

The experiment is conducted using the bug report from the open source software development industry. Different metric suites are used to identify the fault proneness related to it. The software metrics for the object oriented systems using Rhino software with different versions are analyzed. The test case is the multiple versions of a robust, openly available open source software product, Rhino. For the Rhino software, the software development strategy employed is highly iterative, with a bottom-up approach. According to Norris Boyd, creator of the Mozilla Rhino project, Rhino may be considered as an example of the use of the agile software development model in open source software. In the time of the analysis of the metrics, the source code is given as a key to examine the changes from lower version to higher version Hector et al (2007). Here, the Rhino software with diverse versions have examined and analyzed to predict the quality of the software.

5.4.1 Empirical Validation of Selected Quality Metric Suites using Rhino Software

In the research carried out, the three OO metric suites and their capability to predict software quality with the help of the metric measures are

explored. Here, the ability of these three metrics suites to predict the quality for different versions of Rhino, an open-source implementation of JavaScript written in Java is analyzed. To measure these OO metrics, Eclipse plug-in tool is employed. Eclipse is a multi-language software development platform. It has an Integrated Development Environment (IDE) and a plug-in system to extend it. Since, Eclipse has an IDE, it enables to incorporate the data utilized to collect dynamic data and static data in Metrics 1.3.6. Metrics 1.3.6 is an open source plug-in for the Eclipse development environment. By using this static data on software systems can be gathered.

As shown in Figure 5.3, the correlation values obtained for the metric WMC of CK metric suite is tabulated in Table 5.1. The correlation values for the other versions are obtained in the same manner and are affixed in appendix (Appendix 2).

Table 5.1 Correlation Values obtained after Analysis for WMC

Version of Rhino	Correlation Value obtained
rhino15R4	0.199959
rhino15R4_1	0.15069
rhino15R5	0.360823
rhino16R1	0.294248
rhino16R2	0.298223
rhino16R3	0.286427
rhino16R6	0.245933

5.4.1.1 Chidamber and Kemerer Metric Suites vs fault proneness – statistical analysis

In the research, the Descriptive statistics for the Rhino software versions are taken to analyze the software. The statistical results for CK metric suites are used to estimate the quality of the software. The following Table 5.2 illustrates different versions of rhino descriptive statistics for the

CK metric suites. The Table 5.2 shows the statistical results of the Rhino software versions for the chosen metric suites. Within each descriptive statistics, the distribution (mean, median) and variance (standard deviation) of each measure is examined.

Table 5.2 CK Metric Suites of the Rhino Versions

STATISTICAL RESULTS					
Metrics Name		**CK Metric suite**			
Version	**Property**	**WMC**	**DIT**	**NOC**	**LCOM**
Rhino 15R1	**Total**	7153	-	41	-
	Mean	65.027	1.709	0.373	0.314
	Std.Dev	111.744	0.918	1.656	0.368
	Maximum	667	5	16	1.095
Rhino 15R2	**Total**	9821	-	46	-
	Mean	61.381	2.162	0.288	0.361
	Std.Dev	110.749	1.6	1.148	0.373
	Maximum	630	6	11	1.167
Rhino 15R3	**Total**	9931	-	46	-
	Mean	62.069	2.188	0.288	0.355
	Std.Dev	112.239	1.598	1.148	0.368
	Maximum	623	6	11	1.182
Rhino 15R4	**Total**	11131	-	50	-
	Mean	62.534	2.118	0.281	0.339
	Std.Dev	118.529	1.537	1.122	0.363
	Maximum0	670	6	12	1.159
Rhino 15R4_1	**Total**	11151	-	50	-
	Mean	62.646	2.118	0.281	0.339
	Std.Dev	118.456	1.537	1.122	0.363
	Maximum	670	6	12	1.159
Rhino 15R5	**Total**	11729	-	53	-
	Mean	64.445	2.121	0.291	0.341
	Std.Dev	129.899	1.518	1.194	0.367
	Maximum	942	6	12	1.143
Rhino 16R1	**Total**	13154	-	60	-
	Mean	68.51	2.177	0.313	0.354
	Std.Dev	132.631	1.548	1.417	0.376
	Maximum	944	6	17	1.312

Table 5.2 (Continued)

STATISTICAL RESULTS					
Metrics Name		**CK Metric suite**			
Version	**Property**	**WMC**	**DIT**	**NOC**	**LCOM**
Rhino 16R2	**Total**	13181	-	60	-
	Mean	68.651	2.177	0.312	0.356
	Std.Dev	133.554	1.548	1.42	0.375
	Maximum	944	6	17	1.312
Rhino 16R3	**Total**	13283	-	60	-
	Mean	69.182	2.177	0.312	0.356
	Std.Dev	134.119	1.548	1.42	0.375
	Maximum	944	6	17	1.312
Rhino 16R4	**Total**	13283	-	60	-
	Mean	69.182	2.177	0.312	0.356
	Std.Dev	134.119	1.548	1.42	0.375
	Maximum	944	6	17	1.312
Rhino 16R5	**Total**	13283	-	60	-
	Mean	69.182	2.177	0.312	0.356
	Std.Dev	134.119	1.548	1.42	0.375
	Maximum	944	6	17	1.312
Rhino 16R6	**Total**	15382	-	84	-
	Mean	65.735	2.124	0.359	0.34
	Std.Dev	128.137	1.493	1.502	0.367
	Maximum	944	6	19	1.312
Rhino 16R7	**Total**	15388	-	84	-
	Mean	65.761	2.124	0.359	0.34
	Std.Dev	128.142	1.493	1.502	0.367
	Maximum	944	6	19	1.312

5.4.1.2 Robert C. Martin Metric Suites vs fault proneness – statistical analysis

Robert C. Martin Metric Suites is applied to find the statistical results for the quality of the Rhino open source software. The following Table 5.3 illustrates different versions of rhino descriptive statistics for the Robert C. Martin metric suites. The Table 5.3 shows the statistical results of the Rhino software versions for the chosen metric suites. Within each

descriptive statistics, the distribution (mean, median) and variance (standard deviation) of each measure is observed.

Table 5.3 Robert C. Martin metric suites of Rhina Versions

STATISTICAL RESULTS						
Metrics Name		**RC Martin Metric suite**				
Version	**Property**	**Ca**	**Ce**	**In**	**Ab**	**Dn**
Rhino 15R1	**Mean**	4	6.143	0.713	0.029	0.258
	Std.Dev	7.464	7.06	0.34	0.072	0.328
	Maximum	22	21	1	0.205	0.75
Rhino 15R2	**Mean**	4.8	5.6	0.685	0.102	0.239
	Std.Dev	8.773	7.579	0.351	0.239	0.318
	Maximum	30	26	1	0.8	0.833
Rhino 15R3	**Mean**	4.636	4.909	0.678	0.092	0.253
	Std.Dev	8.988	6.288	0.339	0.23	0.306
	Maximum	32	22	1	0.8	0.833
Rhino 15R4	**Mean**	4.818	4.818	0.7	0.111	0.239
	Std.Dev	9.806	5.096	0.323	0.227	0.281
	Maximum	35	18	1	0.8	0.833
Rhino 15R4_1	**Mean**	4.818	4.818	0.7	0.111	0.239
	Std.Dev	9.806	5.096	0.323	0.227	0.281
	Maximum	35	18	1	0.8	0.833
Rhino 15R5	**Mean**	4.909	4.909	0.675	0.129	0.282
	Std.Dev	9.15	5.775	0.321	0.282	0.286
	Maximum	33	21	1	1	0.833
Rhino 16R1	**Mean**	5.786	4.857	0.625	0.167	0.289
	Std.Dev	11.346	5.553	0.351	0.344	0.301
	Maximum	45	22	1	1	0.857
Rhino 16R2	**Mean**	4.438	4.145	0.681	0.146	0.237
	Std.Dev	5.134	5.134	0.336	0.327	0.271
	Maximum	44	21	1	1	0.833
Rhino 16R3	**Mean**	4.5	4.188	0.682	0.15	0.232
	Std.Dev	10.718	5.163	0.337	0.326	0.273
	Maximum	45	21	1	1	0.833

Table 5.3 (Continued)

STATISTICAL RESULTS						
Metrics Name		**RC Martin Metric suite**				
Version	**Property**	**Ca**	**Ce**	**In**	**Ab**	**Dn**
Rhino 16R4	**Mean**	4.5	4.188	0.682	0.15	0.232
	Std.Dev	10.718	5.163	0.337	0.326	0.273
	Maximum	45	21	1	1	0.833
Rhino 16R5	**Mean**	4.5	4.188	0.682	0.15	0.232
	Std.Dev	10.718	5.163	0.337	0.326	0.273
	Maximum	45	21	1	1	0.833
Rhino 16R6	**Mean**	5	4.526	0.688	0.127	0.231
	Std.Dev	12.872	5.614	0.341	0.304	0.276
	Maximum	24	24	1	1	0.833
Rhino 16R7	**Mean**	5	4.526	0.688	0.127	0.231
	Std.Dev	12.872	5.614	0.341	0.304	0.276
	Maximum	58	24	1	1	0.833

5.4.1.3 McCabe's Metric Suites vs fault proneness – statistical analysis

The statistical results for the quality of the Rhino open source software is analyzed with the McCabe's metric suites. The following Table 5.4 enlists different versions of rhino descriptive statistics for the McCabe's Metric Suites. The Table 5.4 shows the statistical results of the Rhino software versions for the chosen metric suites. Within each descriptive statistics, the distribution (mean, median) and variance (standard deviation) of each measure is observed.

Table 5.4 McCabe's metric suites of the rhino Versions

STATISTICAL RESULTS					
Metrics Name		**McCabe Metric suite**			
Version	**Property**	**VG**	**MLOC**	**TLOC**	**NBD**
Rhino 15R1	**Total**	-	22436	29496	-
	Mean	4.814	15.098	-	1.743
	Std.Dev	12.386	43.063	-	1.206
	Maximum	165	670	-	10
Rhino 15R2	**Total**	-	28929	37559	-
	Mean	4.847	14.279	-	1.82
	Std.Dev	12.444	41.793	-	1.252
	Maximum	219	881	-	10
Rhino 15R3	**Total**	-	29238	37975	-
	Mean	4.814	14.173	-	1.834
	Std.Dev	12.247	40.978	-	1.253
	Maximum	210	853	-	10
Rhino 15R4	**Total**	-	31649	40104	-
	Mean	4.943	14.054	-	1.853
	Std.Dev	13.416	41.745	-	1.257
	Maximum	235	974	-	10
Rhino 15R4_1	**Total**	-	31697	40182	-
	Mean	4.936	14.031	-	1.853
	Std.Dev	13.396	41.683	-	1.256
	Maximum	235	974	-	10

Table 5.4 (Continued)

STATISTICAL RESULTS					
Metrics Name		**McCabe Metric suite**			
Version	**Property**	**VG**	**MLOC**	**TLOC**	**NBD**
Rhino 15R5	**Total**	-	33475	42677	-
	Mean	5.056	14.429	-	1.902
	Std.Dev	13.965	42.405	-	1.305
	Maximum	242	1056	-	10
Rhino 16R1	**Total**	-	38901	50054	-
	Mean	4.688	13.864	-	1.892
	Std.Dev	12.755	38.748	-	1.308
	Maximum	281	1230	-	10
Rhino 16R2	**Total**	-	38858	50145	-
	Mean	4.681	13.799	-	1.891
	Std.Dev	12.822	38.21	-	1.31
	Maximum	294	1208	-	10
Rhino 16R3	**Total**	-	39198	50336	-
	Mean	4.666	13.768	-	1.882
	Std.Dev	12.766	38.021	-	1.306
	Maximum	294	1208	-	10
Rhino 16R4	**Total**	-	39198	50336	-
	Mean	4.666	13.768	-	1.882
	Std.Dev	12.766	38.021	-	1.306
	Maximum	294	1208	-	10
Rhino 16R5	**Total**	-	39198	50336	-
	Mean	4.666	13.768	-	1.882
	Std.Dev	12.766	38.021	-	1.306
	Maximum	294	1208	-	10

Table 5.4 (Continued)

STATISTICAL RESULTS					
Metrics Name		**McCabe Metric suite**			
Version	**Property**	**VG**	**MLOC**	**TLOC**	**NBD**
Rhino 16R6	**Total**	-	44315	57594	-
	Mean	4.36	12.561	-	1.831
	Std.Dev	11.929	35.709	-	1.279
	Maximum	305	1265	-	10
Rhino 16R7	**Total**	-	44329	57610	-
	Mean	4.36	12.561	-	1.831
	Std.Dev	11.928	35.704	-	1.278
	Maximum	305	1265	-	10

5.4.2 Data Analysis

The ability of software quality models to accurately identify the critical components allows for the application of focused verification activities ranging from manual inspection to automated formal analysis methods. Software quality models, thus, helps to ensure the reliability of the delivered products. It has become an imperative to develop and apply good software quality models early in the software development life cycle, especially for large-scale development efforts.

The basic hypothesis of software quality prediction is that a module currently under development is fault prone if a module with the similar product or process metrics in an earlier project (or release) developed in the same environment was fault prone. Therefore, the information available early within the current project or from the previous project can be used in making

predictions. The Table 5.5 describes about the packages, classes, methods and defects using the Rhino software versions.

Table 5.5 Characteristics of Rhino software versions used for empirical validation

Rhino versions	No. of Packages	No. of Classes	No. of Methods	Defects Count
Rhino 15R1	7	110	1053	-
Rhino 15R2	10	160	1551	-
Rhino 15R3	11	160	1577	2
Rhino 15R4	11	178	1705	25
Rhino 15R4_1	11	178	1711	12
Rhino 15R5	11	182	1679	152
Rhino 16R1	14	192	2091	125
Rhino 16R2	16	192	2093	143
Rhino 16R3	16	192	2120	77
Rhino 16R4	16	192	2120	1
Rhino 16R5	16	192	2120	-
Rhino 16R6	19	234	2732	73
Rhino 16R7	19	234	2733	2

5.4.3 Correlation Analysis

For each metric measure, the relationship to the defects of the Rhino versions with the software metrics is examined. This is mostly justified by the fact that the defects determine, to some extent, many of its external properties such as fault-proneness or effort. This is to determine empirically whether the measure, even though it is declared as a coupling, classes,

methods, packages or inheritance measure, is essentially measuring its size. For the purpose of analyzing correlations with each metrics suites, the Spearman's rank coefficient between each measure of the chosen metrics and the number of defects present in the Rhino versions is calculated. By using Spearman's rank correlation coefficient technique, the relationships between each explanatory variable (metric) and the dependent variable (the number of defects) are exposed.

Spearman's Rank Correlation: The strength of a link between two sets of data is discovered using the Spearman's Rank Correlation Coefficient. The Spearman's Rank Correlation Coefficient Fieller et al (1957) is an easy correlation coefficient to calculate. This number varies between -1 and +1.

- A correlation coefficient of +1 means perfect positive correlation
- A correlation coefficient close to 0 means no correlation
- A correlation coefficient of -1 means perfect negative correlation.

The Spearman correlation coefficient is often thought of being the Pearson correlation coefficient between the ranked variables. However, in practice, an easier procedure is normally used to calculate ρ. The n raw scores X_i, Y_i are converted to ranks x_i, y_i, and the differences $d_i = x_i - y_i$ between the ranks of each observation on the two variables are calculated by the equation (5.1)

If there are no tied ranks, then ρ is given by:

$$\rho = 1 - \frac{6\sum d_i^2}{n(n^2-1)} \qquad (5.1)$$

where, ρ is the correlation coefficient value, n is the total raw scores, and d_i is the differences between x and y.

If tied ranks exist, Pearson's correlation coefficient between ranks should be used for the calculation:

$$\rho = \frac{\sum_i (x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_i (x_i - \overline{x})^2 \sum_i (y_i - \overline{y})^2}} \quad (5.2)$$

One has to assign the same rank to each of the equal values. It is an average of their positions in the ascending order of the values . The results of the correlation analysis of the selected software metrics suites are given in Table 5.6. Table 5.6 displays the results, which shows that correlation between the values of the chosen metrics and the defects found in those metrics.

Table 5.6 Results of correlation analysis

Metrics		**Rhino 15R4**	**Rhino 15R4_1**	**Rhino 15R5**	**Rhino 16R1**	**Rhino 16R2**	**Rhino 16R3**	**Rhino 16R6**
CK Metrics	WMC	0.2	0.1507	0.3608	0.2942	0.2982	0.2864	0.2459
	NOC	0.135	0.16	0.2791	0.2383	0.1374	0.1205	0.1559
	DIT	0.0959	0.0844	0.2363	0.1897	0.2499	0.2257	0.1277
	LCOM	0.1323	0.0699	0.2239	0.1562	0.093	0.0477	0.1470
	MLOC	0.1878	0.1523	0.3442	0.2896	0.2865	0.2778	0.2407
McCabe Metrics	VG	0.1931	0.1584	0.3619	0.2891	0.2951	0.2788	0.2432
	TLOC	0.1994	0.1741	0.2779	0.2474	0.1826	0.1862	0.208
	NBD	0.1789	0.1389	0.3540	0.2856	0.2919	0.2728	0.2308
RC Martin metrics	A	0.3964	0.2958	0.2326	0.2057	0.0381	0.3677	0.4191
	I	0.1249	0.2922	0.333	0.0993	0.3186	0.2469	0.1132
	Dn	0.1879	0.0567	0.0129	0.2022	0.3281	0.0728	0.1468
	Ca	0.3757	0.0729	0.1871	0.3907	0.1372	0.091	0.2514
	Ce	0.6957	0.6434	0.7969	0.6896	0.6199	0.6659	0.6116

The object oriented metrics are utilized to predict the quality of the object oriented software products. Diverse attributes, which detect the quality of the software, comprise maintainability, defect density, fault proneness, normalized rework and understandability. In this, fault-proneness is an external quality attribute which can be used to predict the fault-prone modules. In the empirical analysis, by analyzing the spearman's rank correlation models across different Rhino versions indicates these metrics may be useful in assessing quality in OO classes produced using modern highly iterative software development processes. It is found that the CK metric suites produce statistical models that are effective in detecting error-prone in classes. It is also found that the methods components in the McCabe metrics suite are good fault-proneness predictors. Also, the RC Martin metric suites are good in predicting the faults in terms of the packages.

5.5 CONCLUSION

Object oriented metrics exist and do provide valuable information to object oriented developers and project managers. A typical empirical validation of object-oriented metrics proceeds by investigating the relationship between each metric and the outcome of interest. The selected three OO metrics suites for their ability to predict software quality are empirically validated. The validated metrics are Chidamber and Kemerer (CK) metrics, Robert C. Martin Suite and McCabe's Metric Suite. The Rhino software is analyzed to predict the software quality with different versions in terms of fault proneness with these metric suites. It is concluded that the CK metric suites produce statistical models that are effective in detecting error-proneness in classes and the method components in the McCabe metric suite are a good fault-proneness predictors and the RC Martin metric suite is good in predicting the faults in terms of the packages.

The empirical analysis aids software professionals to find out the metric suites that can predict faults while developing the metric-quality software products using the OO approach.