

Otto-von-Guericke-University Magdeburg



Department of Computer Science  
Institute for Distributed Systems

## Master Thesis

### **Conception and Prototypical Implementation of a Web Service as an Empirical-based Consulting about Java Technologies**

Author:

Ayaz Farooq

October 12, 2005

Supervised by:

Prof. Dr. habil. Reiner R. Dumke  
Dipl.-Inform. René Braungarten

University of Magdeburg  
Department of Computer Science  
P.O. Box 4120, D-39016 Magdeburg  
Germany



*In the name of Allah, Most Gracious, Most Merciful.*



*To my beloved mother Ghafooran Bibi*

*For my deceased father Wali Muhammad,  
who will always remain alive in my memories*



## **Abstract**

As a fact, the application of object-oriented approach is of high significance in the area of software development since it can abet efficiency or cost effectiveness, and can reduce error probability during software design and implementation. In order to quantify, especially qualitative aspects such as potential error hot spots caused by elevated design complexity, software measurement can strongly assist. Particularly, measurements of Chidamber and Kemerer metrics suite as well as Abreu's MOOD metrics set are presumably most prevalent in practice and provide adequate explanatory power. Especially the object-oriented programming language Java cannot be dismissed from one's thoughts because a lot of Java libraries serve as foundation for contemporary applications. This thesis presents a prototype software measurement web service which performs measurement and evaluation of various Java standard libraries like J2SE, J2EE, J2ME, JWSDP and few others concerning different aspects. The results acquired are beneficial to be used by software designers for aligning and orienting their design with common industry practices. Furthermore, these extensive measurements provide an opportunity for key metrics correlation studies incorporating many thousand Java classes.



## Acknowledgements

I would like to express my gratitude to all those who helped me to complete this thesis.

First, I am indebted to my supervisor Prof. Dr. Reiner R. Dumke who put forward the idea of this thesis work. All the inspiration and motivation behind this work is due to him. I thank him for offering a course on software quality management, specially in english from where I got introduced to the area of software measurement. He proved to be an ideal supervisor during my research.

Special thanks to my advisor Dipl. Inform. René Braungarten who helped me a lot all through my work. His feedback during my work and encouragement is really appreciable. The words are simply not enough to express my gratitude for him. I am also thankful to Dr. Reinhard Koeppel for his help on concepts of object-orientation.

Thanks to Zaheer and Omer for good chitchat which kept me fresh daily for the next days work. I am also thankful to Khazada for careful proof-reading and useful feedback for improving my thesis. I also appreciate Kamran's patience whom I disturbed from time to time for help on Latex.

Thanks to Wikipedia and Google for answering my countless questions every day. Thanks to many open source developers whose works made many of my tasks easier to accomplish.

Finally, I am grateful to my family who gave me moral support during my journey towards higher studies. Special thanks to my mother who always prays for my success, and to my brother Khalid for extending all financial support to me and for always encouraging and pushing me up for hard work.



# Contents

<b>List of Figures</b>	<b>xiii</b>
<b>List of Tables</b>	<b>xv</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Goals of the Proposed Work . . . . .	2
1.3 Thesis Organization . . . . .	3
<b>2 Object-Oriented Design Metrics</b>	<b>5</b>
2.1 Software Measurement . . . . .	5
2.1.1 Metrics and Measures . . . . .	6
2.2 Need for Software Measurement . . . . .	7
2.3 Properties of Software Metrics . . . . .	8
2.4 Object-Oriented Metrics . . . . .	9
2.4.1 Chidamber & Kemerer Metrics Suite . . . . .	9
2.4.2 MOOD Metrics Set . . . . .	11
2.4.3 Importance of Chidamber & Kemerer and MOOD Metrics . . . . .	13
2.5 Related Work . . . . .	14
2.5.1 Theoretical and Empirical Validations . . . . .	14
2.5.2 Size Independence Studies . . . . .	15
2.5.3 Software e-Measurement . . . . .	15
<b>3 Assorted Development Aspects</b>	<b>19</b>
3.1 Problem Definition . . . . .	19
3.2 System Specification . . . . .	20
3.2.1 User's View of Requirements . . . . .	20
3.2.2 Interactive System View . . . . .	22
3.2.3 Activities . . . . .	25
3.3 System Design . . . . .	28
3.3.1 Architectural Strategy . . . . .	28
3.3.2 Physical System Components . . . . .	29
3.4 Implementation . . . . .	31

<b>4 Measurement Exploration</b>	<b>33</b>
4.1 Examined Set of Software	33
4.1.1 Measurement using OOMJ	35
4.2 Applied Statistics	37
4.2.1 Graphical Statistical Methods	37
4.2.2 Numerical Statistical Methods	38
4.3 Measurement Evaluation	43
4.3.1 Size Metrics Results	43
4.3.2 Chidamber & Kemerer Metrics Results	43
4.3.3 MOOD Metrics Results	54
4.4 Synopsis	62
<b>5 Conclusions and Future Work</b>	<b>65</b>
5.1 Thesis Summary	65
5.2 Future Work	66
5.2.1 Validation Studies using the Metrics Results	66
5.2.2 Metrics-Based Analysis of other Competing Technologies	66
5.2.3 Validation of MOOD Metrics	66
5.2.4 Object-Oriented Metrics Database	66
<b>Bibliography</b>	<b>69</b>
<b>A Implementation</b>	<b>75</b>
<b>Declaration</b>	<b>77</b>

# List of Figures

2.1	Measurement Entities in the Three Components of OO-Software . . . . .	6
2.2	A Screen Shot from JMT . . . . .	16
2.3	A Screen Shot from CKJM 1.4 . . . . .	16
3.1	Use-Case Diagram for OOMJ Web Application . . . . .	20
3.2	Metrics Results Sequence Diagram . . . . .	23
3.3	Metrics Analysis Sequence Diagram . . . . .	24
3.4	Evaluate Project Sequence Diagram . . . . .	24
3.5	Download Results Sequence Diagram . . . . .	25
3.6	Activity Diagram: Evaluating a Project . . . . .	26
3.7	Activity Diagram: Analyzing Libraries/Projects . . . . .	27
3.8	Model-View-Controller Architecture . . . . .	28
3.9	Component Diagram for OOMJ Web Application . . . . .	30
3.10	Packages and Classes inside MetricsCalculator Component . . . . .	30
3.11	A Sample XML Result File for MOOD Metrics . . . . .	32
4.1	Title Page from OOMJ . . . . .	36
4.2	C&K Metrics Analysis from OOMJ . . . . .	36
4.3	Example Bar Charts . . . . .	37
4.4	An Example Scatter Plot . . . . .	38
4.5	Examples of Skewness . . . . .	40
4.6	Examples of Kurtosis . . . . .	41
4.7	Bar Charts for C&K Metrics . . . . .	46
4.8	Scatter Charts for C&K Metrics . . . . .	51
4.9	Scatter Charts among selected C&K Metrics . . . . .	53
4.10	Bar Charts for MOOD Metrics . . . . .	56
4.11	Scatter Charts for MOOD Metrics . . . . .	59
4.12	Scatter Charts among selected MOOD Metrics . . . . .	61



# List of Tables

2.1	Empirical Validations of C&K Metrics . . . . .	14
4.1	Size Metrics for Measured Technologies . . . . .	44
4.2	Initial Descriptive Statistics: C&K Metrics for All Libraries . . . . .	45
4.3	Descriptive Statistics: C&K Metrics for J2SE 1.5.0 . . . . .	45
4.4	Descriptive Statistics: C&K Metrics for All Libraries . . . . .	47
4.5	Distribution Test Results for C&K Metrics . . . . .	47
4.6	95% Confidence Interval:Mean of C&K Metrics (all Libraries) . . . . .	48
4.7	Correlation of C&K Metrics with Lines of Code . . . . .	50
4.8	Linear Correlation Coefficient among C&K Metrics . . . . .	54
4.9	Rank Correlation Coefficient among C&K Metrics . . . . .	54
4.10	Descriptive Statistics: MOOD Metrics for J2SE 1.5.0 . . . . .	55
4.11	Descriptive Statistics: MOOD Metrics for all Libraries . . . . .	55
4.12	Distribution Test Results for MOOD Metrics . . . . .	57
4.13	95% Confidence Interval:Mean of MOOD Metrics (all Libraries) . . . . .	57
4.14	Correlation of MOOD Metrics with Size Metrics . . . . .	60
4.15	Linear Correlation Coefficient among MOOD Metrics . . . . .	62
4.16	Rank Correlation Coefficient among MOOD Metrics . . . . .	62
4.17	C&K Metrics Heuristics . . . . .	63
4.18	MOOD Metrics Heuristics . . . . .	63



---

## List of Acronyms

AHF	Attribute Hiding Factor
AIF	Attribute Inheritance Factor
API	Applications Programming Interface
BCEL	Byte Code Engineering Library
C&K	Chidamber and Kemerer
CAME	Computer Assisted Software Measurement and Evaluation
CASE	Computer-Aided Software Engineering
CBO	Coupling between Objects
CEWOLF	Chart Enabling Web Object Framework
CKJM	Chidamber and Kemerer Java Metrics
CMMI	Capability Maturity Model Integration
COF	Coupling Factor
DIT	Depth of Inheritance Tree
DOM	Document Object Model
ERP	Enterprise Resource Planning
ISBSG	International Software Benchmarking Standards Group
IT	Information Technology
J2EE	Java 2 Platform, Enterprise Edition
J2ME	Java 2 Platform, Micro Edition
J2SE	Java 2 Platform, Standard Edition
JAR	Java Archive
JB	Jarque-Bera
JSP	JavaServer Pages
JWSDP	Java Web Services Developer Pack
KS	Kolmogorov-Smirnov
LCOM	Lack of Cohesion in Methods
LOC	Lines of Code
MHF	Method Hiding Factor
MIF	Method Inheritance Factor
MOOD	Metrics for Object-Oriented Design
MVC	Model-View-Controller
NASA/SEL	National Aeronautics and Space Administration/Software Engineering Laboratory
NOC	Number of Children
OO	Object-Oriented
OOMJ	Object-Oriented Measurement of Java Technologies
POF	Polymorphism Factor
RFC	Response for Class
SAX	Simple API for XML
UML	Unified Modeling Language
WMC	Weighted Method per Class
XML	Extensible Markup Language
XPath	XML Path Language



# 1 Introduction

*The skill of writing is to create a context  
in which other people can think.*  
–Edwin Schlossberg

Computer systems have become integral part of our everyday life and the world without computers is beyond our imagination now. Software running inside those computer systems is itself of paramount importance. Software engineering means the application of a systematic, disciplined and quantifiable approach to the development, operation, and maintenance of software. As we are becoming more and more reliant on software systems, more effort is being put on development of efficient, reliable and cost-effective software. A technique devised to yield manageable, scalable and reliable software is object-oriented paradigm. An object-oriented software consists of independent cooperating objects, each object being capable of receiving messages, processing data, and sending messages to other objects.

In recent years there has been growing awareness about the important role of measurement in software engineering. Software measurement has evolved into a key software engineering discipline. It is now considered to be a basic software engineering practice, as evidenced by its inclusion in the Level 2 maturity requirements of the Software Engineering Institute's Capability Maturity Model Integration (CMMI)<sup>1</sup>. Software measurement helps us understand software, manage a software project and improve the software process itself.

## 1.1 Motivation

With the ever increasing number of software projects and concern for sophistication, size and complexity of software systems is also increasing. There are many horrifying stories of software crashes. Recent examples are failure of FBI's Virtual Case File system and Hudson Bay Company's (Canada) inventory system [1]. As some organizations waste tens of billions of dollars annually on failed software projects, the key to consistently creating large, reliable, and efficient IT systems remains elusive. Why do software projects fail at all? Reasons, among others, are inability to handle project's complexity and sloppy development practices.

Software complexity and casual development practices are a rich source of failure, and they can cause errors at any stage of an IT project. Many software development processes do not prevent bugs; they merely put off dealing with them until the end. We need a proactive

---

<sup>1</sup>[www.sei.cmu.edu/cmmi](http://www.sei.cmu.edu/cmmi)

preventive approach right from the beginning of the design and development activity. In this regard, software measurement helps us manage and control a software project through all of its phases.

More specifically, talking about an object-oriented software design, practitioners and designers need a quantified and experience-based view on how to make best use of object-oriented mechanisms inline with the time tested practices being undertaken in the software industry. An empirical analysis on the implementation of object-oriented paradigm in some standard libraries can reveal some of the common industry practices. Once designers have such data, they can use it for benchmarking their own application designs and can highlight possible problem areas. Object-oriented software metrics provide such quantitative view about the implementation of object-oriented constructs in software design.

Chidamber & Kemerer metrics suite [2] is probably most widely referenced set of object oriented metrics. Abreu's MOOD (Metrics for Object-Oriented Design) [3] is a system level metrics set which quantifies the use of four main mechanisms of object-oriented design that is *encapsulation*, *inheritance*, *polymorphism* and *message-passing*. Since Java is a common object-oriented programming language, a metrics based measurement and analysis of Java libraries can reveal particular design trends that can be considered as guidelines for other Java-based user applications. Using this information, practitioners and designers can flag outlier program components which can be potential problem areas.

This thesis measures and analyzes a vast set of Java standard libraries such as those provided by J2SE, J2ME, JWSDP, Java Card, few Jakarta Apache projects as well as several other open source libraries. The prototype measurement service also provides facility of similar analysis of any user application for metrics based comparison against standard Java libraries. The analysis provides cognitive design heuristics that can lead to a consistent development approach handling the vital complexity issues.

## 1.2 Goals of the Proposed Work

In view of the above discussion, following is the brief outline of the goals set forth for this thesis work.

- Determination of the metrics values about various Java technologies (such as J2SE, J2EE and other related libraries) based on the object-oriented metrics of Chidamber & Kemerer and Abreu.
- Prototypical implementation of a an online object-oriented Java metrics measurement and analysis service.
- Provision of a means of technology based experiences founding a basis of a scalable concept for the chosen Java technologies as a consulting service.
- Exploration of relationships among the metrics in question based on analysis of empirical metrics data set.

## 1.3 Thesis Organization

The remaining part of this thesis is organized as follows.

### Chapter 2

This chapter sets the foundation of the thesis. It introduces the concept of software measurement and its broad application areas. The metrics concerning this thesis work are discussed. Detailed explanation of how these metrics apply to Java programming concepts is also part of this chapter. The chapter ends with a review of related research work and legacy metrics collection/analysis experiments and applications.

### Chapter 3

This chapter describes the proposed approach. The properties and constraints of an effective online software measurement and analysis application are presented here. The application's development process is explained in detail from different view points using a number of diagrams from UML<sup>2</sup> 2.0 (a common design language used today).

### Chapter 4

This chapter is the heart of the thesis in that it extracts helpful metrics heuristics for software metrics in question within this thesis work. The chapter analyzes the metrics results using a number of statistical techniques. Interesting relationships between system size and the calculated metrics are explored. In addition, correlations among those metrics themselves are derived, too.

### Chapter 5

This chapter concludes the thesis and describes the potential areas of future work.

---

<sup>2</sup><http://www.uml.org/>



## 2 Object-Oriented Design Metrics

*Not everything that can be counted counts,  
and not everything that counts can be counted.*  
–Albert Einstein

Whenever we talk about evaluating quality of something we intuitively visualize a numerical view of its characteristics thereby judging its quality by comparing these characteristics with those that we encountered earlier, that is, its size, price, color etc. In fact, we are then thinking in metrics. The idea of metrics also applies to quality of software. This chapter briefly describes the main concept behind this thesis, that is software metrics or more specifically object-oriented software metrics.

Section 2.1 introduces the notion of software measurement and different measurement scenarios. Section 2.2 explains significance of software measurement in a software engineering process. Characteristics of software metrics are listed in section 2.3. The object-oriented design metrics applicable to this work are described in section 2.4 along with metrics bindings for Java. Finally, section 2.5 presents relevant research work done in this field of empirical software engineering.

### 2.1 Software Measurement

Measurement, in general, is the process by which numbers or symbols are assigned to attributes of entities in the real world in such a way as to describe them according to clearly defined unambiguous rules. Software measurement applies to a software engineering process thereby measuring numerous entities encountered along the way. According to Dumke [4], software measurement is directed to three main components in the object-oriented software development (see also Fenton [5]):

- the **process measurement** for understanding, evaluation and improvement of the development method,
- the **product measurement** for the quantification of the product (quality) characteristics and validation of these measures,
- the **resource measurement** for the evaluation of the supports (CASE tools, measurement tools etc.) and the chosen implementation system.

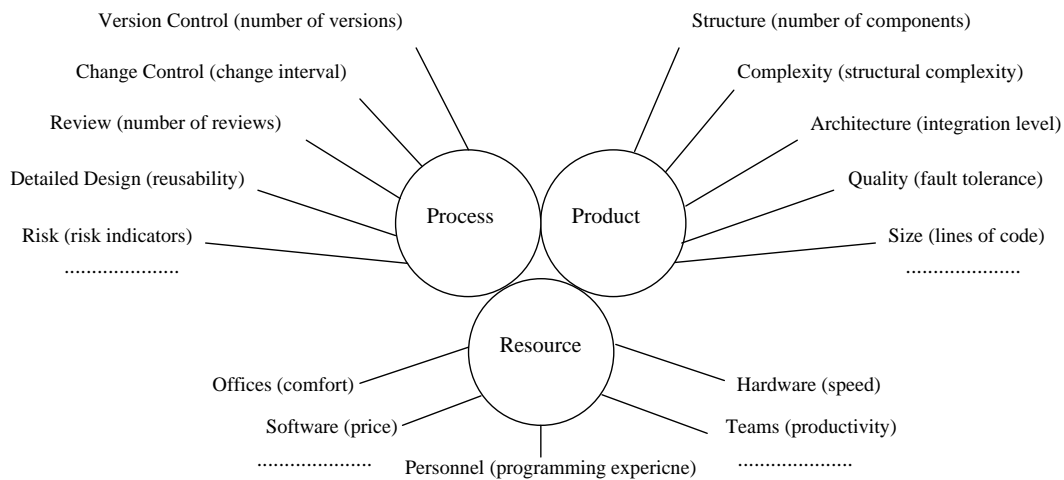


Figure 2.1: Measurement Entities in the Three Components of OO-Software

Figure 2.1 shows examples of some of the measurement attributes within these areas.

Seen from another viewpoint, attributes belonging to these three aspects of software (that is, process, product and resource) can fundamentally be of two types:

- **Internal attributes** are those that can directly and independently be measured in terms of product, process or resource itself. For example, size of a software can directly be measured by measuring lines of code, number of modules, number of classes or methods etc. Effort, time, algorithmic complexity and coupling are other examples of internal attributes. These attributes are mostly of concern to software architects and designers. Certain sets of internal attributes are the focus of this thesis.
- **External attributes** are features of a software product, process or resource which are externally visible to people and environment such as quality, usability, reliability, cost and stability. For example, poor usability of a software is visible to the users if the user interface is complicated. External attributes are mostly of interest to managers and users of a software system.

Usually, internal attributes alone are not much useful unless we consider them jointly with external attributes exploring some possible relationship between the two. Thus, one of the goals of a software measurement and analysis study is to develop a prediction model to derive external attributes from internal attributes by applying a theoretical or empirical validation.

### 2.1.1 Metrics and Measures

There has been confusion among the software engineering community surrounding the terms *metrics* and *measures*. Some researchers differentiate between the two while others use both these terms synonymously.

A metric is a quantitative measure of the degree to which a system, component, or process possesses a given attribute<sup>1</sup>. Dumke [6] and Zuse [7] mention that a software metric, in accordance with measure theory, is a distance function which assigns numbers to the attributes of software components. Fenton [8] states that a metric is a function  $m$  defined on pairs of objects  $x$  and  $y$  such that  $m(x, y)$  represents a distance between  $x$  and  $y$ . Such metrics must satisfy certain properties:

- $m(x, x) = 0$  for all  $x$ : that is, the distance from a point to itself is 0;
- $m(x, y) = m(y, x)$  for all  $x$  and  $y$ : that is, the distance from  $x$  to  $y$  is the same as the distance from  $y$  to  $x$ ;
- $m(x, z) < m(x, y) + m(y, z)$  for all  $x, y$  and  $z$ : that is, the distance from  $x$  to  $z$  is no larger than the distance measured by stopping through an intermediate point.

According to IEEE<sup>2</sup>, measure means to ascertain or appraise by comparing to a standard. According to Dumke [6] and Zuse's [7] definition a software measure, in accordance with measurement theory, evaluates (or makes measurable) a software attribute with the help of a scale and a scale unit. In Fenton's view [8] measurement is a mapping from empirical world to the formal, relational world. Consequently, a measure is the number or symbol assigned to an entity by this mapping in order to characterize an attribute. In another paper [5] Fenton mentions that a measure is an empirical objective assignment of a number (or symbol) to an entity to characterize a specific attribute.

From above definitions it may be inferred that a metric is an observable value (color of skin) while measure associates meaning to that value by adding human judgement (beauty).

## 2.2 Need for Software Measurement

There has been enough evidence that many large software projects fail even before deployment or become too complex to maintain. To name a few are Virtual Case File system of FBI, ERP system of Avis Europe PLC (UK) and supply-chain management system of J Sainsbury PLC (UK) [1]. Consequently, heavy cost and effort is expended for maintenance tasks. This calls for an effective prediction, control, assessment and estimation of various aspects of software development process and product. Software measurement comes out to help us perform such activities efficiently.

Discussing the applications of software measurement for managers, Shepperd [9] points out that it helps them to determine staffing requirements early on in a project, monitor software reliability over time and assess maintainability of ageing software. Importance of software measurement is also stressed by the by inclusion of a measurement and analysis process

---

<sup>1</sup>IEEE Standard Glossary of Software Engineering Terminology, IEEE Std 610.12 1990

<sup>2</sup>IEEE Standard Glossary of Software Engineering Terminology, IEEE Std 729 1983

area in Software Engineering Institute's Capability Maturity Model Integration (CMMI) and other related standards.

Amongst others, software measurement assists software designers in the software development process by enabling them to resolve such issues as estimating:

- size of the software product early on in the design phase
- time and effort needed to develop a software product
- error-proneness of the intended software and potential error hot spots
- resources needed for an effective development process
- level of maintainability of software product from early design documents
- complexity of the developed code even when developer has not yet started writing any code
- testability of software by quantifying structural design diagrams

### 2.3 Properties of Software Metrics

A number of software metrics have been defined in literature yet not all of them have been proved to be enough significant. Therefore, some fundamental principles and characteristics have to be considered while defining a new software metric. Zuse [7], Henderson-Sellers [10] and Gustafson & Prasad [11] provide a comprehensive overview of different metrics properties proposed by researchers. Conte et al. [12] suggest that a metric should be objective; that is, the value must be computed in a precise manner. Henderson-Sellers [10] says that a metric must be at least valid, reliable and practical. It must also be intuitive (have face validity) and possess some form of internal validity. Abreu [3] proposes following criteria for assessing suitability of a software metric:

1. Metrics determination should be formally defined.
2. Non-size metrics should be system size independent.
3. Metrics should be dimensionless or expressed in some consistent unit system.
4. Metrics should be obtainable early in the life-cycle.
5. Metrics should be down-scaleable.
6. Metrics should be easily computable.
7. Metrics should be language independent.

## 2.4 Object-Oriented Metrics

Several object-oriented design metrics have been proposed by researchers [2, 3, 13, 14, 15, 16, 10, 17]. These metrics describe diverse aspects of an object-oriented design. This work involves two popular sets of object-oriented measures namely *Chidamber & Kemerer* (herein after C&K) metrics suite and *Metrics for Object-Oriented Design* (herein after MOOD) proposed by F.B. Abreu. It is evident that definition of these metrics is not language independent. This thesis evaluates and analyzes these metrics for Java technologies. Since various OO programming languages differ slightly in implementation of some of the language constructs, we need to adapt these metrics definitions according to concepts of programming language in question. Therefore, an overview of these metrics along with Java bindings is presented in the next two sections.

### 2.4.1 Chidamber & Kemerer Metrics Suite

This metrics suite was proposed in [2] by S. R. Chidamber and C. F. Kemerer. The six structural design metrics proposed by them are explained here.

- Weighted Method per Class (*WMC*)

It is sum of complexities of all methods in a class. Consider a class  $C_1$  with methods  $M_1, \dots, M_n$  that are defined in the class. Let  $c_1, \dots, c_n$  be complexities of each of these methods. Then WMC has been defined as:

$$WMC = \sum_{i=1}^n c_i$$

For this work, complexity of each method is assumed to be unity and so WMC is simply sum of all defined methods.

**C&K-Java Binding:** This work considers WMC as count of all defined methods inside a class with any access modifier. This does not include inherited and abstract methods. This is because inherited methods do not actually belong to this class. Abstract methods do not have a body and so no complexity measure is possible for them.

- Depth of Inheritance Tree (*DIT*)

Depth of inheritance of the class is the DIT metric for the class.

**C&K-Java Binding:** This study takes DIT as the maximum length of the inheritance tree up to the root. A class may implement an interface and that interface may extend one or more interfaces. Those interfaces are counted as well towards DIT measure.

- Number of Children (*NOC*)

Number of immediate sub-classes subordinated to a class in class hierarchy.

**C&K-Java Binding:** It is the number of immediate sub-classes of a class. For an interface it is the number of classes implementing it plus number of other interfaces extending this interface.

- Coupling between Objects (*CBO*)

CBO for a class is count of the number of other classes to which it is coupled. Two classes are coupled together if methods of one use methods or instance variables of other. Excessive coupling between object classes is detrimental to modular design and prevents reuse. The more independent a class is, the easier it is to reuse it in another application.

**C&K-Java Binding:** A class can call methods from another class either through inheritance or using an object of the other class. CBO should measure both forms of these couplings.

- Response for a Class (*RFC*)

$RFC = |RS|$  where *RS* is response set for the class. This is a set of methods that can potentially be executed in response to a message received by an object of that class. Since it specifically includes methods called from outside the class, it is also a measure of the potential communication between the class and other classes.

**C&K-Java Binding:** This includes all defined and inherited methods inside this class plus methods called on objects of other classes in any method of this class.

- Lack of Cohesion in Methods (*LCOM*)

The LCOM is a count of the number of method pairs whose similarity is 0 minus the count of method pairs whose similarity is not zero. The degree of similarity for two methods  $M_1$  and  $M_2$  in class  $C_1$  is given by:

$\sigma() = \{I_1\} \cap \{I_2\}$  where  $\{I_1\}$  and  $\{I_2\}$  are the sets of instance variables used by  $M_1$  and  $M_2$

The larger the number of similar methods, the more cohesive the class, which is consistent with traditional notions of cohesion that measure the inter-relatedness between portions of a program. A high cohesion is favored in class designs.

**C&K-Java Binding:** Instance variables are the ones with any access modifier.

## 2.4.2 MOOD Metrics Set

F. B. Abreu proposed these system-level metrics in [18]. This set of six metrics measures four main structural mechanisms of object-oriented design that is *encapsulation* (Method Hiding Factor and Attribute Hiding Factor), *inheritance* (Method Inheritance Factor and Attribute Inheritance Factor), *polymorphism* (Polymorphism Factor) and *message-passing* (Coupling Factor). An explanation of the metrics with Java bindings follows except for coupling factor which was not measured.

**Common Java Binding Note:** This set of metrics applies to system level. While measuring these metrics for standard libraries like J2SE etc. a top level Java package is assumed to constitute one *system*. For example, *java.awt* will be considered as one system and other packages stemming from it like *java.awt.event* and *java.awt.color* will be considered part of *java.awt* system.

- Method Hiding Factor (*MHF*)

$$MHF = \frac{\sum_{i=1}^{TC} \sum_{m=1}^{M_d(C_i)} (1 - V(M_{mi}))}{\sum_{i=1}^{TC} M_d(C_i)}$$

where:

$$V(M_{mi}) = \frac{\sum_{j=1}^{TC} is\_visible(M_{mi}, C_j)}{TC - 1}$$

and:

$$is\_visible(M_{mi}, C_j) = \begin{cases} 1 & \text{iff } j \neq i \text{ and } C_j \text{ may} \\ & \text{call } M_{mi} \\ 0 & \text{otherwise} \end{cases}$$

The numerator is the sum of invisibility of all methods defined in all classes. The invisibility of a method is the percentage of the total classes from which this method is not visible. The denominator is the total number of methods defined in the system under consideration.

### MOOD-Java Binding:

*TC*– total number of classes in the system/package

*M<sub>d</sub>(C<sub>i</sub>)*– number of constructors and methods defined with any access modifier excluding abstract and inherited methods

- Attribute Hiding Factor (*AHF*)

$$AHF = \frac{\sum_{i=1}^{TC} A_d(C_i) \sum_{m=1}^{TC} (1 - V(A_{mi}))}{\sum_{i=1}^{TC} A_d(C_i)}$$

where:

$$V(A_{mi}) = \frac{\sum_{j=1}^{TC} is\_visible(A_{mi}, C_j)}{TC - 1}$$

and:

$$is\_visible(A_{mi}, C_j) = \begin{cases} 1 & \text{iff } j \neq i \text{ and } C_j \text{ may} \\ & \text{reference } A_{mi} \\ 0 & \text{otherwise} \end{cases}$$

*AHF* is defined similar to *MHF*

**MOOD-Java Binding:**

$A_d(C_i)$ – number of all attributes with any access modifier but not including inherited ones

- Method Inheritance Factor (*MIF*)

$$MIF = \frac{\sum_{i=1}^{TC} M_i(C_i)}{\sum_{i=1}^{TC} M_a(C_i)}$$

where  $M_a(C_i) = M_d(C_i) + M_i(C_i)$

The numerator is the sum of inherited methods in all classes of the system. The denominator is the total number of available methods in all classes.

**MOOD-Java Binding:**

$M_i(C_i)$ – number of inherited methods but not overridden

$M_d(C_i)$ – number of defined non-abstract methods with any access modifier.

$M_a(C_i)$ – number of methods that class  $C_i$  can call

- Attribute Inheritance Factor (*AIF*)

$$AIF = \frac{\sum_{i=1}^{TC} A_i(C_i)}{\sum_{i=1}^{TC} A_a(C_i)}$$

where  $A_a(C_i) = A_d(C_i) + A_i(C_i)$

It is defined analogous to *MIF*.

**MOOD-Java Binding:**

$A_i(C_i)$ – number of inherited attributed

$A_d(C_i)$ – number of defined attributes with any access modifier.

$A_a(C_i)$ – number of attributes that class  $C_i$  can reference

- Polymorphism Factor (*POF*)

$$POF = \frac{\sum_{i=1}^{TC} M_o(C_i)}{\sum_{i=1}^{TC} [M_n(C_i) \times DC(C_i)]}$$

where  $M_d(C_i) = M_n(C_i) + M_o(C_i)$

The numerator is the number of methods that override inherited methods. The denominator is maximum number of possible distinct polymorphic situations. Thus, it is an indirect measure of the relative amount of dynamic binding in a system.

**MOOD-Java Binding:**

$DC(C_i)$  number of direct and indirect sub-classes of this class in the current system

$M_o(C_i)$ – number of overriding methods

$M_n(C_i)$ – number of new defined non-overriding methods with any access modifier

### 2.4.3 Importance of Chidamber & Kemerer and MOOD Metrics

Among the many object-oriented software metrics defined so far, C&K and MOOD metrics are two fundamentals sets of OO design measures.

C&K metrics set is the most widely referenced in software engineering literature. Since its definition by Chidamber and Kemerer in 1994, this metrics set has been a focus of many empirical validations by researchers. These metrics have been tested for relationship with external attributes of software like fault proneness, maintainability, productivity and rework effort.

MOOD metrics set is unique in being a system level metrics set. These metrics describe level of implementation of key mechanisms of object-oriented design. These metrics have also

been validated in few studies mainly by the author himself.

## 2.5 Related Work

Research in the area of software measurement has been in focus within the software engineering community over the past many years. A review of the related research work is being presented next.

### 2.5.1 Theoretical and Empirical Validations

An overwhelming majority of researchers have concentrated on theoretical and empirical validation of object-oriented software metrics. These two types of validations are used to demonstrate respectively that: (a) a measure is really measuring the attribute it is purporting to measure, and (b) the measure is useful in the sense that it is related to other variables in expected ways (as defined in the theories)[19].

Chidamber and Kemerer metrics have been empirically validated in many studies. An overview of few of these studies is given in table 2.1. For a comprehensive summary of empirical studies of object-oriented metrics see [20].

Table 2.1: Empirical Validations of C&K Metrics

Reference	Internal Metrics	Dependent Variable
[21]	CBO & RFC	Change Proneness of Classes
[22]	All C&K	Fault Proneness
[23]	DIT	Modifiability and Understandability
[24]	All C&K	Software Defects
[25]	DIT, NOC, CBO	Fault Proneness
[26]	DIT, NOC, CBO, RFC	Fault Proneness
[27]	DIT	Maintenance
[28]	All C&K	Productivity, Rework Effort
[29]	All C&K	Fault Proneness
[30]	All C&K	Fault Proneness
[31]	All C&K	Fault Proneness
[32]	CBO	Ripple Effects in OO Systems
[33]	All Except LCOM	Fault Proneness of Classes

In view of high number of empirical validations of C&K metrics and their importance as good quality indicators, this thesis investigates into these metrics evaluations for a number of Java libraries. Benlarbi et. al. [34] are of the view that no thresholds exist for C&K metrics. This

work, too, does not suggest any cut-off values for these measures but presents only a view of most prevalent industry trends and practices.

In contrast, Abreu's MOOD metrics have not been much thoroughly empirically validated. He himself studied relationship of these metrics with defect density and rework [18]. In another work [35] he derived some design heuristics for MOOD metrics using a collection of C++ libraries. In [36] he analyzed these metrics for some Eiffel libraries and again derived some design heuristics. This thesis evaluates these metrics for very large set of Java standard libraries and projects and presents some design heuristics. Harrison et al. [37] provided a theoretical validation of MOOD metrics set. Mayer and Hall [38] presented a critical analysis of this metrics set and opined some flaws in MOOD definitions.

### 2.5.2 Size Independence Studies

Discussing about desirable properties of software metrics, Abreu [3] suggests that non-size metrics should be system size independent. Few studies have tested both C&K and MOOD metrics for their correlation to some size metrics.

Briand et al. [31] studied C&K metrics for correlation against class size using few C++ libraries as data set. Another work [30] again derived some relationship of these metrics with class size. Xiao [39] tested a subset of C&K metrics for correlation to class size. The MOOD metrics have been examined for correlation with some system size property in [35] and [36]. These studies were performed on a small data set consisting of only a couple of classes. This thesis replicates the C&K and MOOD metrics correlation studies with a much bigger sample size (many thousand Java classes) and results thus obtained can be considered more reliable. Abreu himself [36] emphasizes a need for correlation studies of his metrics with a larger sample size.

### 2.5.3 Software e-Measurement

Software e-measurement refers to web based applications developed to support software measurement processes, activities and communities. Dumke [40] proposes the notion of computer assisted software measurement and evaluation (CAME) tools for identifying all kinds of metrics tools in the software life cycle. In their book on software measurement, Dumke et al. ([41], Chapter 4) provide a comprehensive list of various web and non-web based software measurement applications.

- JMT

Java Measurement Tool (JMT) was developed at University of Magdeburg, Germany. It calculates C&K metrics for Java source code and presents results in the form of simple graphs and tabular structure. Figure 2.2 shows a screen shot from this tool.

- CKJM

Chidamber and Kemerer Java Metrics (CKJM) was developed by Prof. Spinellis of

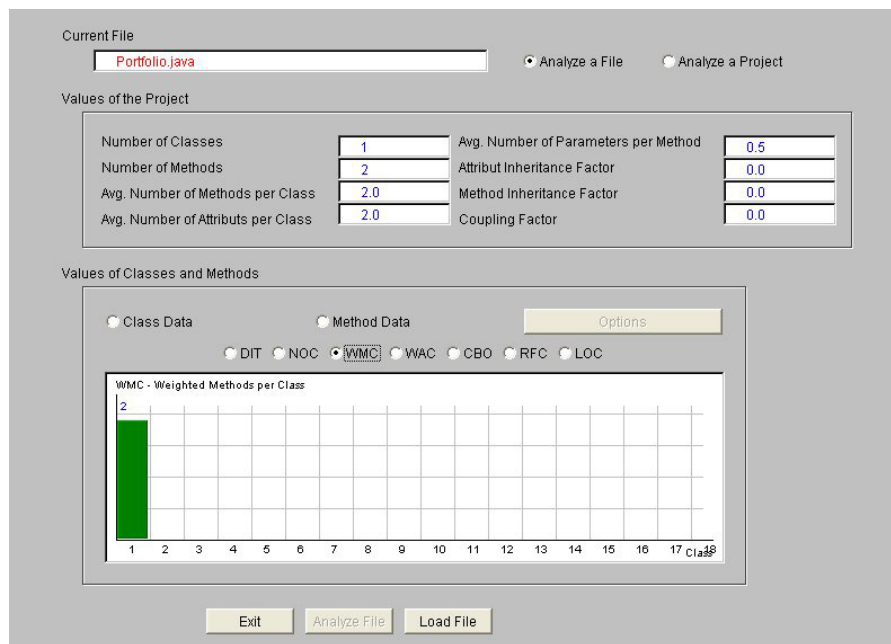


Figure 2.2: A Screen Shot from JMT

Athens University of Economics and Business, Greece. The current command line version is 1.4 which also supports metrics results output in the form of XML files. Figure 2.3 shows a screen shot from this tool.

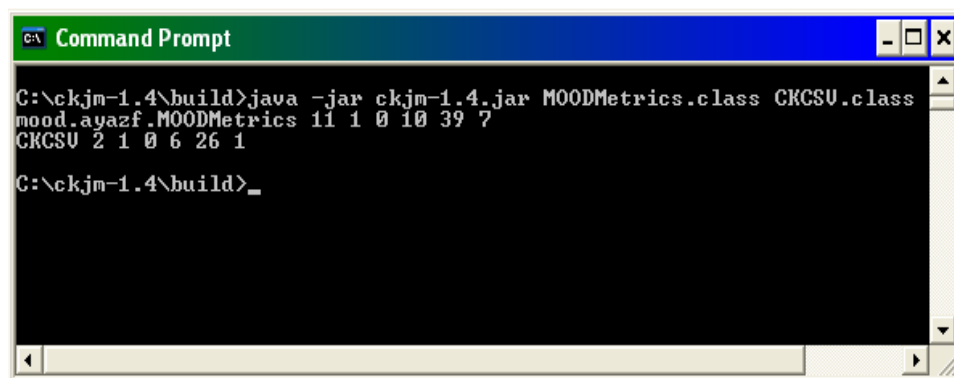


Figure 2.3: A Screen Shot from CKJM 1.4

- **LOGISCOPE**  
It is a powerful program evaluation tool which supports code measurement including quality evaluation of large-scale software. Among many other metrics it supports collection of C&K and MOOD metrics for C++ and Java source code.
- **JMetric**

It is another metrics tool for Java intended to provide metrics based project analysis to the practitioner. It supports only a subset of the C&K metrics.

Except LOGISCOPE which is a commercial software, the mentioned tools only provide numbers without adequate graphical and otherwise analysis of results. Moreover, *all* of these projects do not provide any metrics results storage facility for metrics based comparison among different libraries and projects, thereby providing an experience based metrics heuristics. Compared with these, this thesis work combines calculation, storage and analysis of metrics results among a lot of libraries and projects. A step further, it also attempts to derive metrics heuristics as well as discover statistical relationship among the calculated metrics.



## 3 Assorted Development Aspects

*Perfection (in design) is achieved not when there is nothing more to add,  
but rather when there is nothing more to take away.*  
–Antoine de Saint-Exupéry

This chapter provides a an architectural overview of the software measurement application called *Object-Oriented Measurements of Java technologies* (OOMJ), using a number of different design views, to depict different aspects of the system. This overview is intended to capture and convey the significant architectural decisions that have been made on the system.

Section 3.1 briefly describes problem definition. Section 3.2 explains system specification by presenting different views of the system. Design architecture and physical components of the system are discussed in section 3.3. Finally, section 3.4 deals with implementation and programming strategies employed to develop this application.

### 3.1 Problem Definition

The OOMJ is a software measurement web application envisaged to provide a quantitative analysis of Java technology libraries and applications. This web application attempts to fulfil following goals:

- Provide a measurement view of Java technologies based on Chidamber & Kemerer Metrics
- Provide a measurement view of Java technologies using MOOD Metrics set
- Enable users to quantitatively analyze Java technology libraries
- Enable users to measure and analyze online any Java library and/or application
- Provide measurement results in a portable form for further customized view and analysis

Unified modeling language (UML 2.0) is mainly a diagrammatic notation for modeling systems using object-oriented concepts [42]. Several different types of UML 2.0 diagrams are used in analysis, design and implementation phases of a software project. A few of these diagrams being used here to outline steps taken in design and implementation of this measurement and evaluation web application.

## 3.2 System Specification

This section specifies requirements of the proposed web application. Use case, sequence and activity diagrams are being used to elaborate various aspects of system functionalities.

### 3.2.1 User's View of Requirements

Use case diagrams are a useful easy-to-understand preliminary step in defining the requirements of a system from user's point of view. A use case diagram consists of *actors* who are actual external users of the system and *use cases* which refer to distinct functionalities provided by the system to the user. A little stick-figure is used as an icon for actor while an ellipse symbolizes a use case. Figure 3.1 represents a use case diagram showing main functions of the OOMJ application.

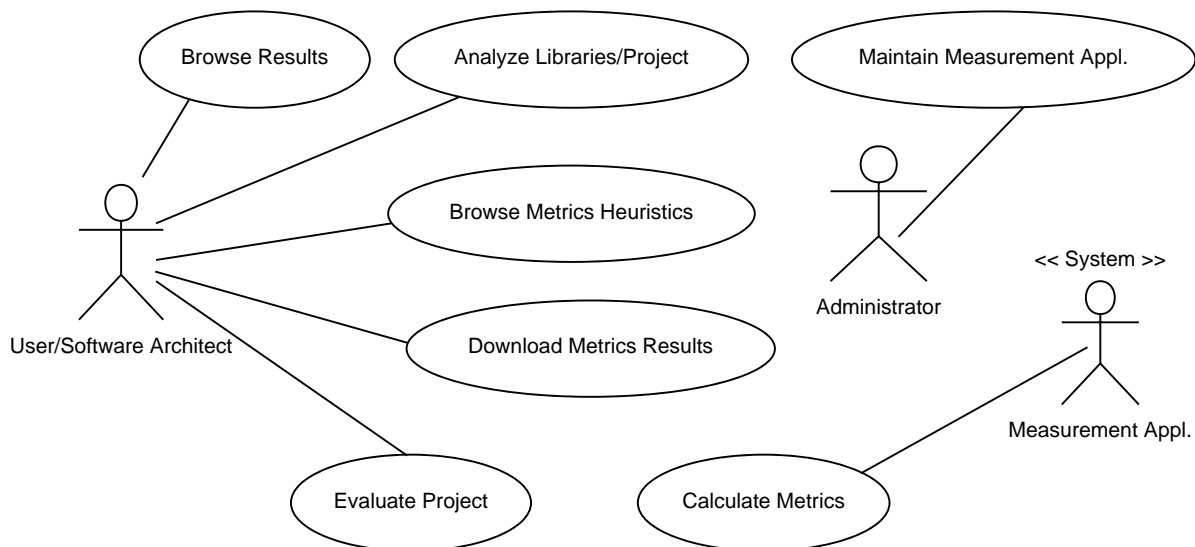


Figure 3.1: Use-Case Diagram for OOMJ Web Application

A more detailed textual specification of the main use cases shown in figure 3.1 is given next.

**Use Case: Browse Results**

Actors: User/Designer

Purpose: Provide a tabular view of metrics results

Overview: The user accesses the web application and views summary of metrics results for various Java libraries and projects.

Typical Course of Events:

1. This use case begins when the user starts the web application and chooses to view metrics results.

2. The user selects type of metrics and technology/application for results.
3. The system displays metrics results in tabular form.

**Use Case: Analyze Libraries/Project**

Actors: User/Designer

Purpose: Provide a graphical analysis of metrics results

Overview: The user accesses the web application and views different types of graphs illustrating metrics results for various Java libraries and projects.

Typical Course of Events:

1. This use case begins when the user starts the web application and chooses to view metrics analysis.
2. The actor selects type of metrics
3. The system displays measured technologies/applications and types of graph available for analysis.
4. The actors selects appropriate technologies and graph type.
5. The system displays the desired graph.

**Use Case: Evaluate Project**

Actors: User/Designer

Purpose: Provide a metrics evaluation for user projects

Overview: The designer or user accesses the web application and submits his own Java project for online evaluation. The system evaluates the application and provides the metrics results.

Typical Course of Events:

1. The user selects to submit his own project for online evaluation.
2. The system provides a page for uploading project files.
3. The actor uploads his core application and additionally used library files for evaluation.
4. The system calculates the desired metrics and provides results for online view and analysis.

Typical Course of Events:

**Use Case: Download Metrics Results**

Actors: User/Designer

Purpose: Provide metrics results in portable form for further evaluation.

Overview: The user accesses the web application and selects different technologies/applications for viewing results. The system provides metrics results for download in an XML file format.

1. The actor selects type of metrics and technology/application for downloading metrics results.
2. The system provides a link to download metrics result files in XML format.
3. The actor downloads the result files for further analysis.
4. Alternatively, the actor can also first evaluate his project online and then download metrics results.

**Use Case: Calculate Metrics**

Actors: Measurement Application

Purpose: Calculate metrics results for the provided project

Overview: When the web application receives a project for online evaluation, it calls measurement component which calculates metrics values and stores results in XML files.

Typical Course of Events:

1. The web application receives a Java project from user for an online calculation of specified metrics.
2. The measurement application acquires the project files in particular locations.
3. The measurement component then calculates metrics values and stores results in XML files in a predefined data repository.
4. The web application can either browse through those results or can use them for generating graphical analysis.

### 3.2.2 Interactive System View

Now that we have got a primary view of system requirements evident from the use case diagram and description of few significant use cases, we need to further explore system behavior for precise definition of object interactions to be carried out to implement functionalities delineated by those use cases.

System behavior is a description of *what* a system does, without explaining *how* it does it. One part of that description is a system *sequence diagram* [42]. A sequence diagram specifies collaboration among objects ordered over time, usually describing behavior of a single use case. A sequence diagram consists of *objects* represented by named rectangles, *messages* among those objects by lines with arrow heads and *time* by a vertical lifeline extending below each object. Sequence diagrams corresponding to few of the main use cases described in the previous section are being illustrated in figure 3.2 to 3.5.

Figure 3.2 details the steps followed in use case *Browse Results*. The 'Controller' object processes user's request for viewing metrics results and creates a 'MetricsResults' object. The user then selects libraries/projects and types of metrics from the page generated by 'MetricsResults'. The results are extracted from XML metrics data files and displayed to the user.

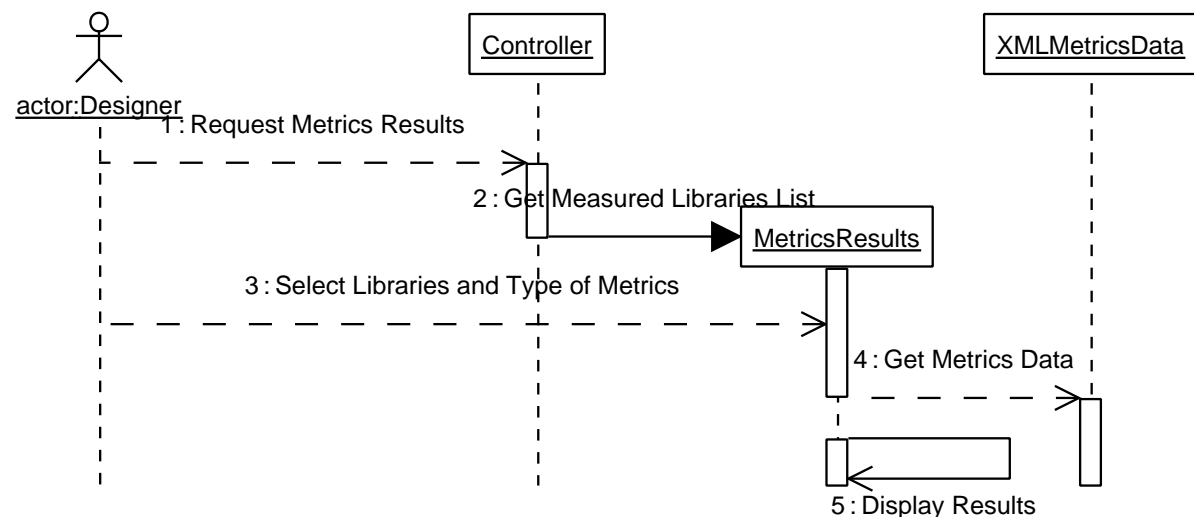


Figure 3.2: Metrics Results Sequence Diagram

The 'Analyze Libraries/Projects' use case is further illustrated in figure 3.3. The 'Controller' object dispatches user's request for results analysis to 'GraphAnalysis' object. The user then selects appropriate options from among the list of libraries, metrics and types of available graphs. The 'GraphAnalysis' then gets metrics data from XML result files and generates desired graphs.

The sequence diagram in figure 3.4 elaborates steps taken during an online evaluation of user-provided Java projects. As usual, the 'Controller' object routes user's request for a project evaluation to 'EvaluateProject'. The user then uploads his relevant project files to the 'EvaluateProject' object. This object then calls 'CalculateMetrics' which calculates the desired metrics and renders the results in XML files.

Figure 3.5 enumerates object interactions for downloading metrics results by users. The 'MetricsResults' object displays available technologies and metrics. The user then selects desired result files which he can then download from a link provided by 'MetricsResults' object.

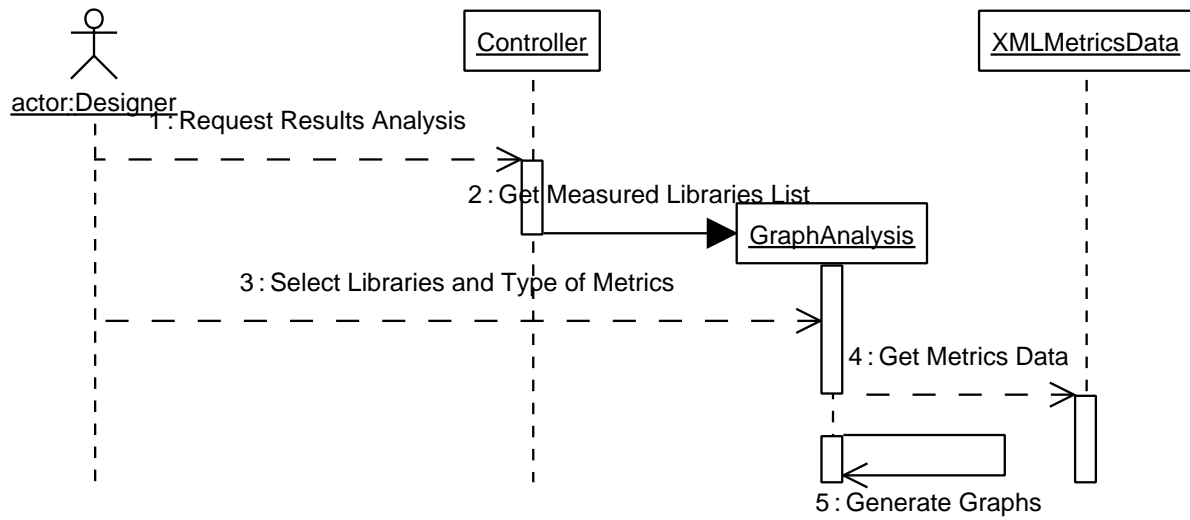


Figure 3.3: Metrics Analysis Sequence Diagram

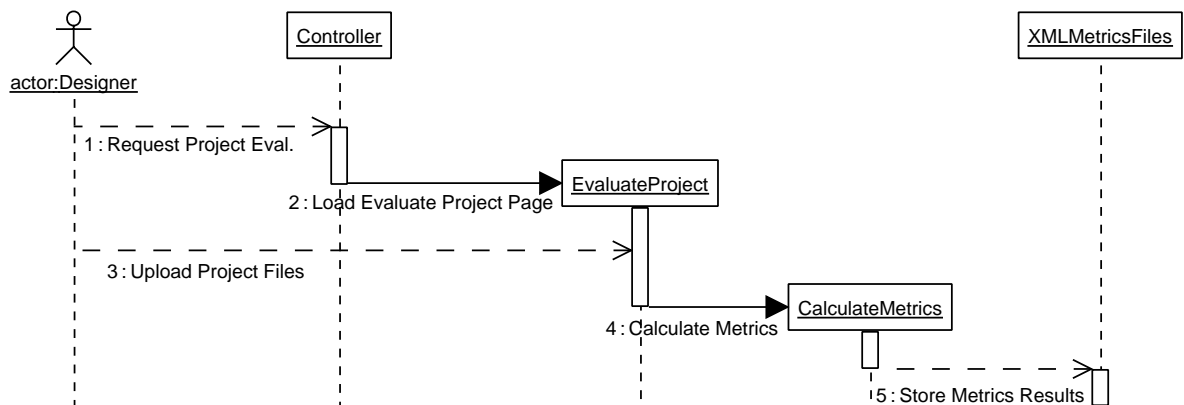


Figure 3.4: Evaluate Project Sequence Diagram

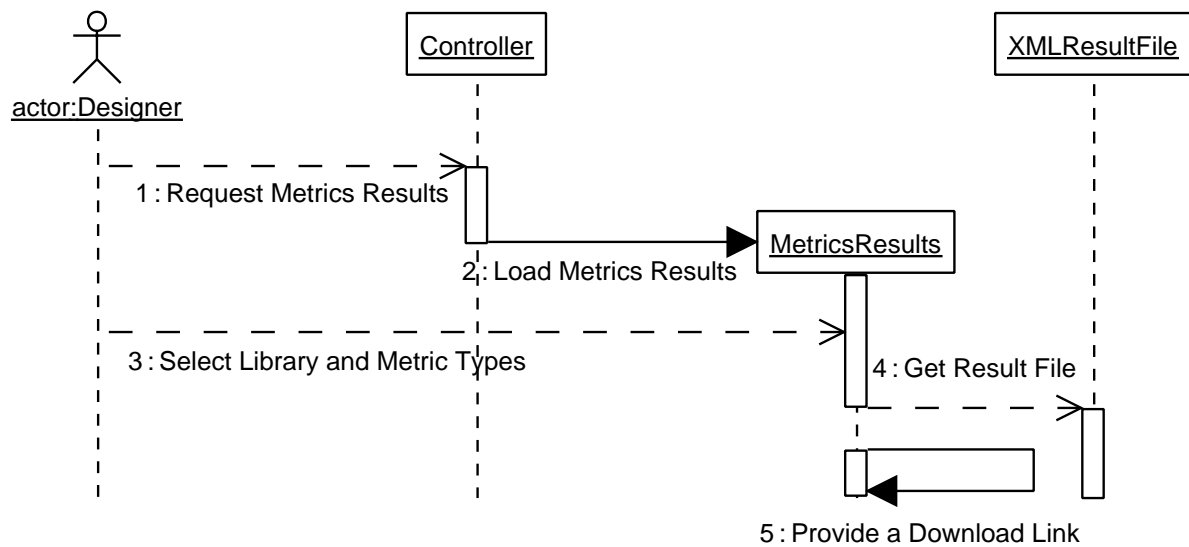


Figure 3.5: Download Results Sequence Diagram

### 3.2.3 Activities

Yet another way to present system specification is through numerous activities taking place during the user interactions with the system. *Activity diagrams* prescribed in UML 2.0 illustrate flow of activities allowing conditional and parallel behavior. An activity diagram is similar to a flow chart commonly used for describing programming logic. These are drawn corresponding to each of the use cases defined in the specification. A few activity diagrams are being given here.

Figure 3.6 models the logic of the 'Evaluate Project' use case. A vertical line partitions this diagram into two parts called *swim lanes*. These partitions refer to actors-in this diagram to user and system. This diagram starts with a filled circle followed by rounded rectangles representing *activities*. A diamond symbol represents a *decision* with the conditions on outgoing decision edges. At the end of the diagram we observe a thick filled horizontal line where output paths from two activities join. It is called a *fork*. The bull's eye symbol marks the end of activity.

Similarly, figure 3.7 refers to activities undertaken in 'Analyze Libraries/Project' use case. The diagram is quite simple and self explanatory.

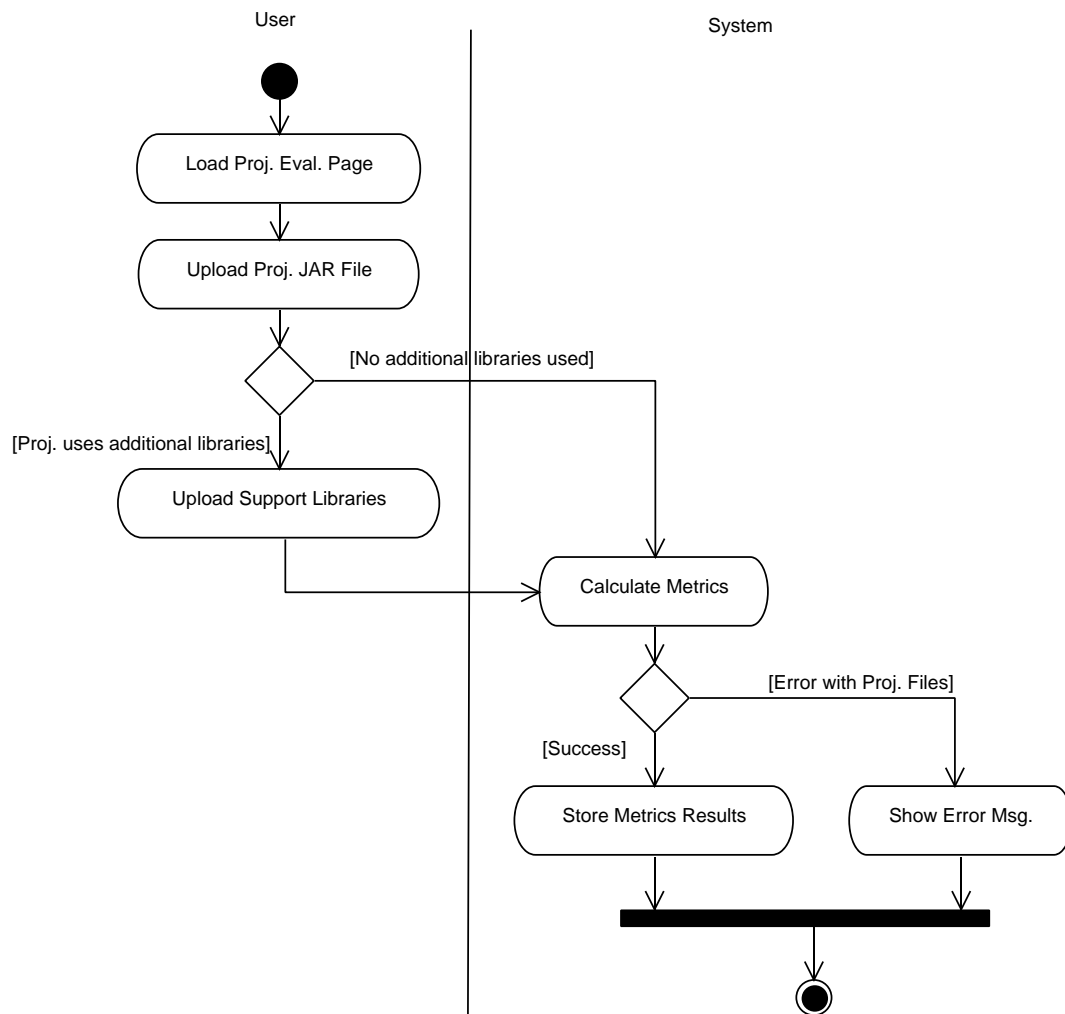


Figure 3.6: Activity Diagram: Evaluating a Project

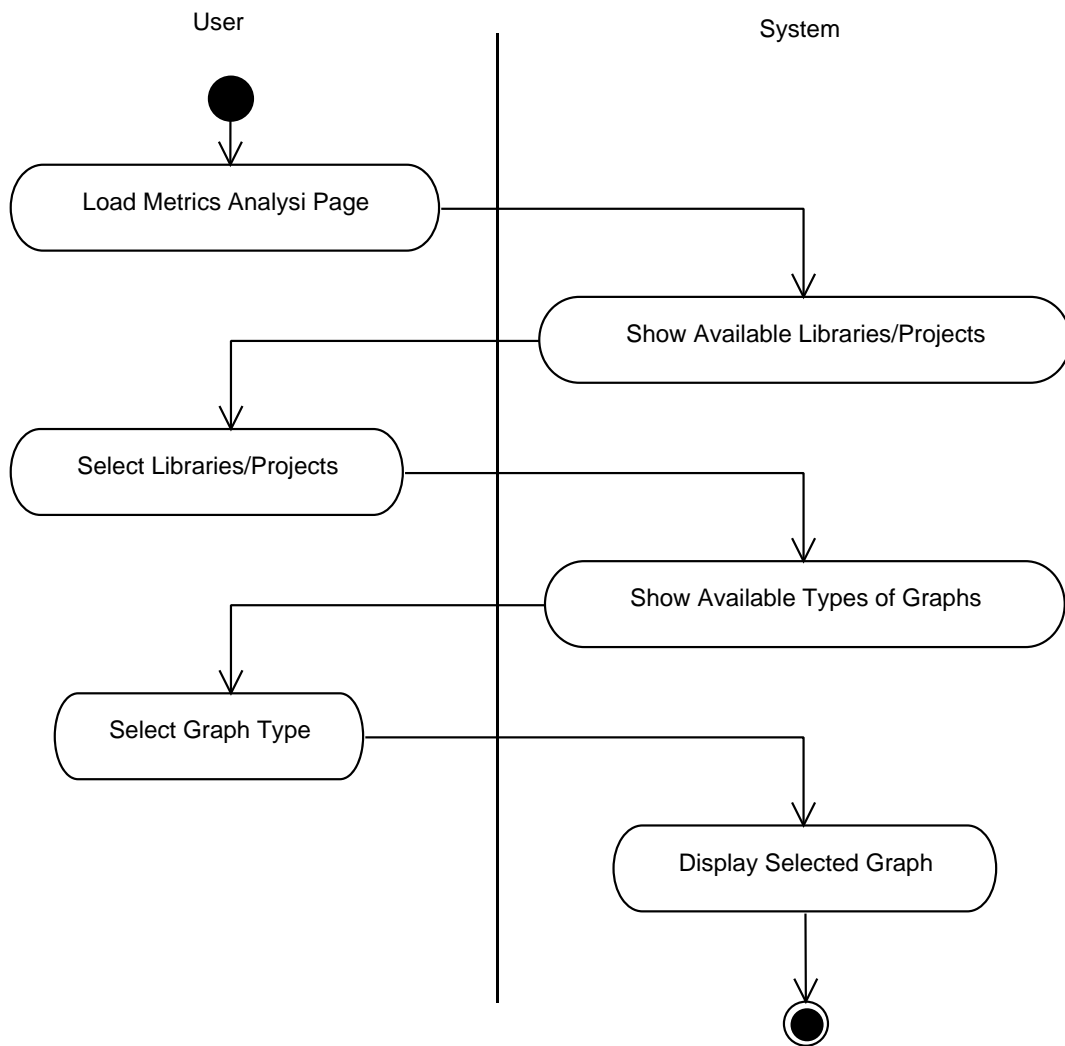


Figure 3.7: Activity Diagram: Analyzing Libraries/Projects

### 3.3 System Design

This section provides an overview of how system functionalities and responsibilities have been assigned to various components and also explains interdependency among those components. It describes adopted design pattern and component diagrams forming the basis of this application.

#### 3.3.1 Architectural Strategy

OOMJ is based on *model-view-controller* (MVC) architecture. An MVC based application consists of three distinct components to handle presentation, control logic and business/data model separately. An overview of MVC architecture is given in figure 3.8.

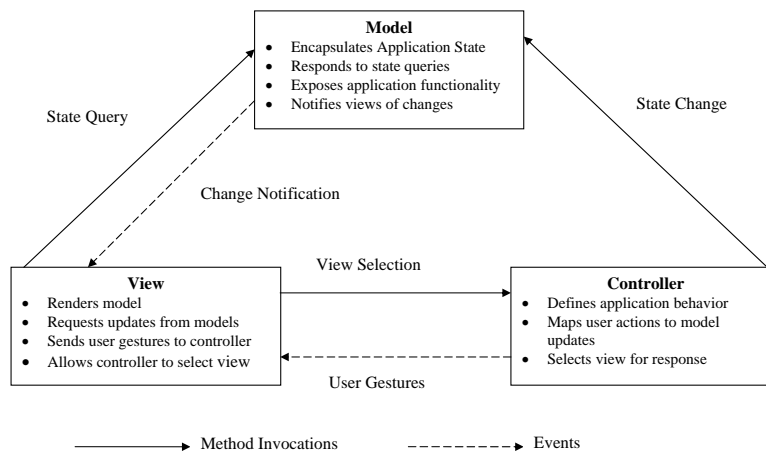


Figure 3.8: Model-View-Controller Architecture

It divides functionality among objects involved in maintaining and presenting data to minimize the degree of coupling between the objects [43]. This strategy facilitates design of interactive web applications by providing a flexible framework.

Within the framework of MVC, this web application implements J2EE design pattern named *Front Controller*. A design pattern is a set of techniques devised to solve a familiar problem. Front controller pattern provides a single centralized controller which receives all incoming client requests, forwards each request to an appropriate request handler, and presents an appropriate response to the client.

A description of MVC components and Front Controller pattern implementation for this application is given next.

- **Model:** It contains the data and business logic components. In a J2EE application, model may consist of simple Java Beans or Enterprise Java Beans (EJBs). This web application uses simple Java Beans components.

- **View:** It makes up presentation of the application and may consist of several html/JSP pages thereby accessing data managed by *model*. There may be multiple views in an application. A combination of html and JSP pages is being employed in this application as presentation components.
- **Controller:** As the name suggests, it interacts with the client capturing events and updates the model. This application contains a single controller implemented as a Java Servlet.

The *controller* interacts with the user, captures events and appropriately changes *model* or *view*. *Model* then performs internal processing and updates *view* which is then presented to the user. These components can be designed independently and an internal change in one of them does not affect other components.

### 3.3.2 Physical System Components

Physical diagrams show dependency among the software components making up the complete application. These dependencies show how a change in one component can cause a possible change in the dependant component. A component usually consists of one or more class files providing a definite system functionality.

Figure 3.9 shows component diagram for the OOMJ web application. The core components with lists of implementing classes are being discussed below.

- **Metrics Calculator**

This is the principal metrics measurement component developed under this work and is the main 'workhorse' behind this web application. It calculates C&K and MOOD metrics for any Java library or project provided in the form of a JAR file. It further consists of three packages as shown in figure 3.11.

*mood.ayazf.utils* package contains useful utility classes for calculating some primary object-oriented metrics which will be used to calculate C&K and MOOD metrics. Package *ck.ayazf* contains classes which calculate some of C&K metrics. Rest of the metrics calculations are borrowed from *ckjm 1.0*, an open source Java metrics tool. Classes calculating MOOD metrics are contained in *mood.ayazf* package.
- **Results Extractor**

One of the functions of this web application is to enable a tabular view of metrics results for various libraries and projects. This component generates and runs a query to extract metrics values from XML files when requested by the web application. *DOM4J 1.6.1* open source library is being used by this component to send XPath queries to XML files.
- **Graphs Generator**

This component draws several different types of graphs into the web pages for visual

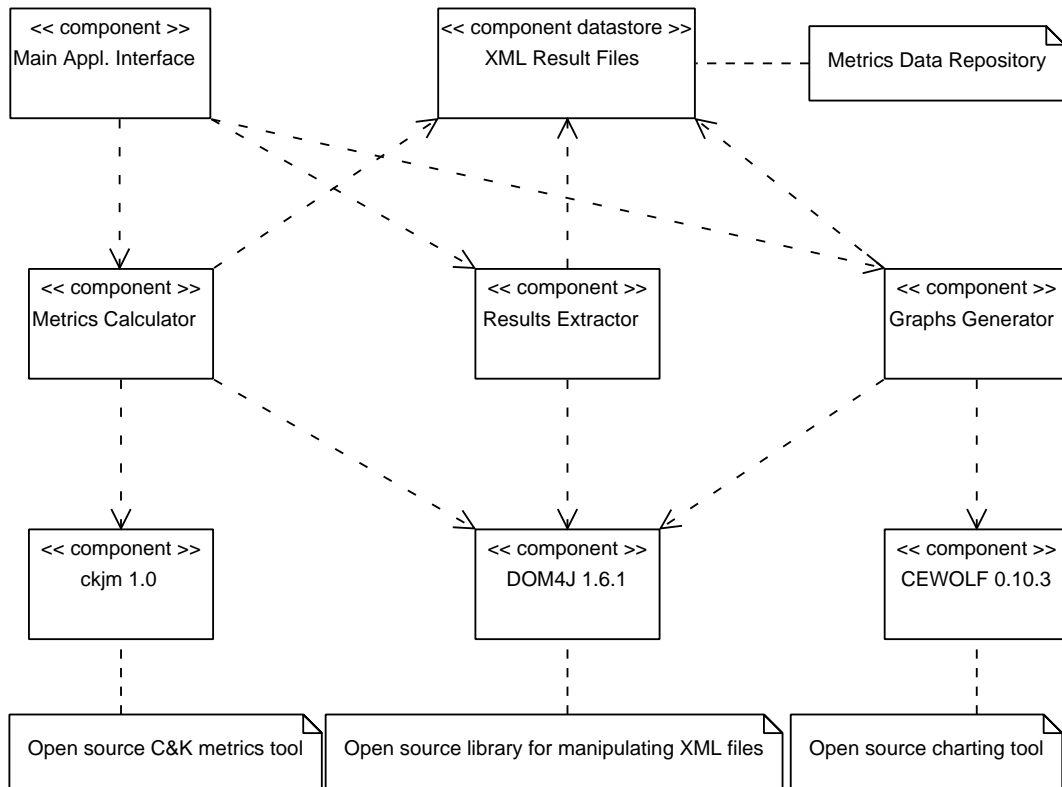


Figure 3.9: Component Diagram for OOMJ Web Application

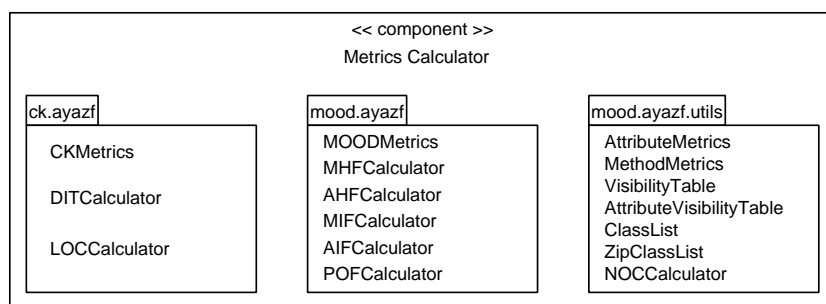


Figure 3.10: Packages and Classes inside MetricsCalculator Component

analysis of metrics results. The three classes comprising this component are *MetricsValueUtility*, *CKGraphsData* and *MOODGraphsData*. CEWOLF 0.10.3, another open source tool for creating charts is being used to draw graphs. The classes within this component first extract desired metrics values from XML files and then use CEWOLF to generate different graphs.

## 3.4 Implementation

This section presents different implementation aspects of this web application. It describes various programming strategies and design decisions taken to solve many programming problems encountered during course of development of this software.

### 1. Core Metrics Calculation

At the core of this measurement application lies the metrics calculation component which calculates Chidamber & Kemerer and MOOD metrics for any given Java library or project. J2SE's package *java.lang.reflect* and Apache BCEL 5.1 (Byte Code Engineering Library) provide several useful classes for calculating some very preliminary object-oriented metrics. Although *java.lang.reflect* package is less simpler than Apache BCEL, yet it is far more powerful to provide many mechanisms needed to calculate required metrics. For example, it is very vital to know declaring class of a method to correctly identify inherited, overriding and declared methods. Apache BCEL does not provide this facility which is a very critical information needed to calculate MOOD metrics. Moreover, access modifiers of methods and attributes can easily be identified using *java.lang.reflect* as compared to BCEL.

### 2. Choice for Metrics Results Data Format

Metrics results data has to be stored for further review and analysis and download by a variety of users. Some relational data structure or XML are two possible choices. Although the metrics results data set is enough structured to be put in a relational database, yet due to high concern for portability and ease of manipulation, XML format has been preferred.

### 3. Parsing XML Result Files

Extensive manipulation of metrics result files is needed for tabular display of metrics data as well as for construction of various graphs. There is a concern for a simple but efficient Java based library for handling XML files. DOM4J 1.6.1 has been chosen for it. It provides both SAX and DOM based approach for accessing XML files. It has been used first for creating, storing and then accessing those XML metrics result files for tabular display of data or to be used in graphs. Furthermore, the choice of some simple XML query language is also essential. XPath has been chosen because of simplicity of queries and keeping in view the simple structure of XML result files.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--MOOD Metrics Results for the dom4j technology (code author:afarooq)-->
-<dom4j>
  <!--dom4j element contains metrics results (in %) for main packages.-->
  -<dom4j-1.6.1>
    <MHF>99.051994</MHF>
    <AHF>98.81668</AHF>
    <MIF>58.125984</MIF>
    <AIF>66.26596</AIF>
    <POF>3.8874753</POF>
  </dom4j-1.6.1>
  -<dom4j-1.5.2>
    <MHF>99.084885</MHF>
    <AHF>98.86888</AHF>
    <MIF>57.97147</MIF>
    <AIF>65.91928</AIF>
    <POF>3.8825355</POF>
  </dom4j-1.5.2>
-</dom4j>
```

Figure 3.11: A Sample XML Result File for MOOD Metrics

## 4 Measurement Exploration

*There are two possible outcomes: if the result confirms the hypothesis, then you've made a measurement. If the result is contrary to the hypothesis, then you've made a discovery.*  
–Enrico Fermi

Performing an accurate empirical data collection experiment is as important as effectively analyzing and interpreting the data hence obtained. Usually, collecting huge empirical data consumes much of researcher's time and a relatively small effort is expended on extracting meaningful results out of that experiment. This chapter summarizes the results of extensive measurements performed on several standard Java libraries as well as few open source Java applications and projects.

Section 4.1 opens the chapter with detail of set of libraries examined in this thesis work. Section 4.2 briefly introduces various numerical and graphical statistical techniques that have been employed to analyze the metrics results. Section 4.3 deals with analysis of metrics evaluation for the Chidamber & Kemerer and MOOD metrics. The last section 4.4 is devoted to a short summary of results enveloping metrics heuristics and metrics correlation analysis derived in section 4.3.

### 4.1 Examined Set of Software

Below is a short description of set of libraries and other applications used for this measurement work.

- J2SE 1.5.0  
Java 2 Platform, Standard Edition 1.5.0 is the currently available version of standard application programming interfaces (APIs/classes) serving as a foundation for applications developed in Java. It provides a complete environment for applications development on desktops and servers and for deployment in embedded environments.
- J2EE SDK 1.4, 1.3.1 & 1.2.1  
The Java 2 Platform, Enterprise Edition (J2EE) defines the standard for developing multi-tier enterprise applications. The J2EE platform provides a complete set of services to application components, and handles many details of application behavior automatically, without complex programming.

The current version is J2EE 1.4 SDK while older versions were J2EE 1.3.1 and 1.2.1. These three versions were measured so as to be able to compare changes over different versions of J2EE libraries.

- JWSDP 1.5

The Java Web Services Developer Pack (Java WSDP) is a free integrated toolkit that can be used to build, test and deploy XML applications, Web services, and Web applications with the latest Web service technologies and standard implementations. This study measures version 1.5 which, among others, contains following main components

- XML and Web Services Security v1.0
- Java Architecture for XML Binding(JAXB) v1.0.3 FCS
- Java API for XML Processing (JAXP) v1.2.6 FCS
- SOAP with Attachments API for Java (SAAJ) v1.2.1 FCS
- and few others

- Java Card 2.2.1

Java Card technology provides a secure environment for applications that run on smart cards and other devices with very limited memory and processing capabilities. Applications written in the Java programming language can be executed securely on cards from different vendors.

- J2ME (Mobile/Wireless)

The Micro Edition of the Java 2 Platform provides an application environment that specifically addresses the needs of commodities in the vast and rapidly growing consumer and embedded space, including mobile phones, pagers, personal digital assistants, set-top boxes, and vehicle telematics systems.

- dom4j 1.6.1 & 1.5.2

It is set of open source Java libraries allowing manipulation of XML files and working with XPath and XSLT. It supports parsing of XML files using JAXP, and serial (SAX) and tree based (DOM) access to those files.

- Apache Tomcat 5.5.9 & 4.1.31

Apache Tomcat is the servlet container that is used in the official reference implementation for the Java Servlet and JavaServer Pages technologies. The Java Servlet and JavaServer Pages specifications are developed by Sun Microsystems. Tomcat is the most widely used web server for testing web applications at development time.

- Apache BCEL 5.1

Byte code engineering library (BCEL) is an open source project from Apache. It provides a convenient way to create, analyze and manipulate Java class (binary) files. It works similar to *java.lang.reflect* package of J2SE.

- Apache Cactus 1.7  
It is another open source project from Apache. It is a simple test framework for unit testing server-side java code, for example Servlets, EJBs, Tag Libs and Filters.
- Apache Lucene 1.4.3  
Lucene is a text search engine library written entirely in Java. It is a technology suitable for nearly any application that requires full-text search, especially cross-platform.
- JFreeChart 1.0.0  
JFreeChart is a useful open source free library for creating various kinds of charts and graphs including scatter, line, bar, area and many others.
- Cewolf 0.10.3  
It is another Java library which can be used inside Java Servlets and JavaServer Pages to create many different kinds of graphs in web pages. It uses JFreeChart as its core.
- HtmlUnit 1.6  
HtmlUnit is a java unit testing framework for testing web based applications. It is also freely available and is open source.

Set of library files (Java classes) performing related or similar function, are grouped inside a Java *package*, or folder in general terms. Neither all packages in these libraries have been measured nor inner classes have been entertained due to implementation complexities.

#### 4.1.1 Measurement using OOMJ

OOMJ (Object-Oriented measurement of Java Technologies) is the software measurement web service (discussed in chapter 3) designed to reliably compute Chidamber & Kemerer and MOOD metrics for any Java library or application and provide an efficient graphical analysis of the obtained results. It is unique in being freely available online and providing the metrics results in portable XML format for further customized analysis or otherwise use. This application has been used to evaluate Chidamber & Kemerer and MOOD metrics for the Java libraries discussed in section 4.1. Figures 4.1 and 4.2 provide a few screen shorts from this application.

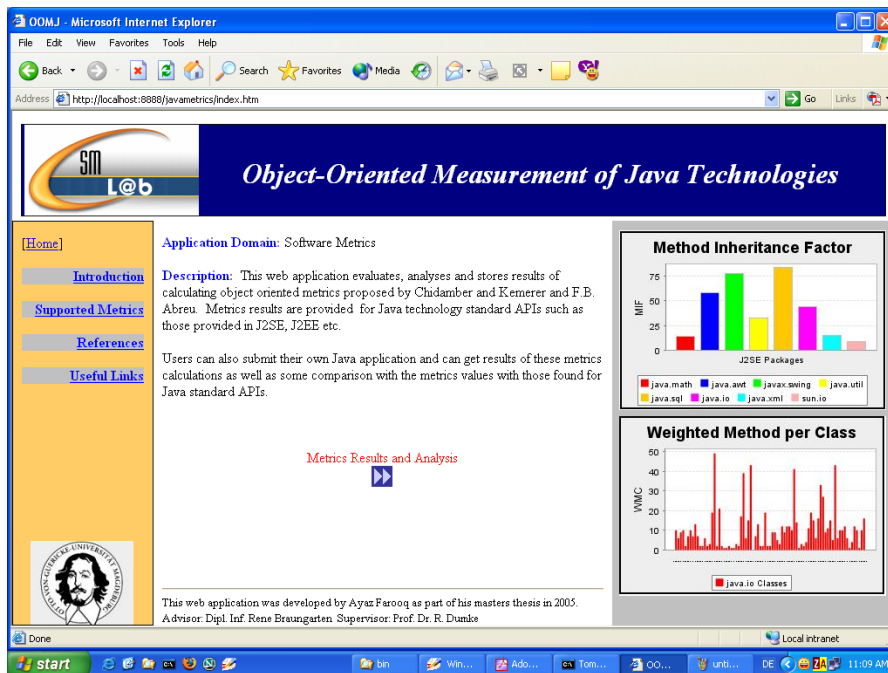


Figure 4.1: Title Page from OOMJ

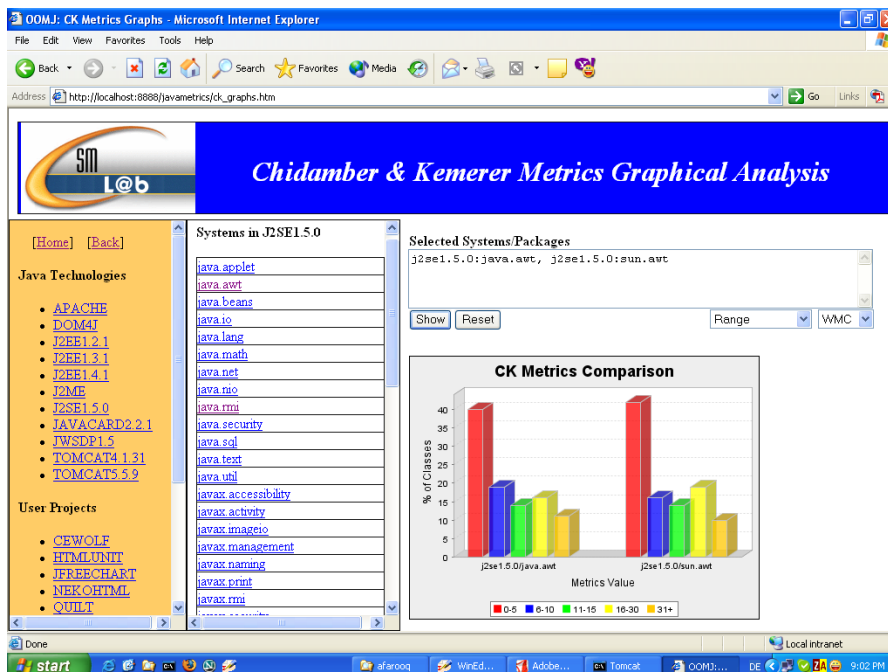


Figure 4.2: C&K Metrics Analysis from OOMJ

## 4.2 Applied Statistics

Data collected as a result of an experiment themselves only supply little useful information. They have to be processed somehow to be converted into meaningful information. Statistics provides us various methods of analyzing the raw data from different view points. One can then derive some results from such analysis. A brief introduction to a number of statistical methods is being given here. These methods will later be used in analysis of the metrics results.

### 4.2.1 Graphical Statistical Methods

A picture speaks louder than many numbers. Graphical statistical methods are an initial step towards getting insight into the evaluated data. Using a number of techniques like bar chart, histogram, scatter plot, and line graph, one can gain a useful initial interpretation of the concerned data. Two of these techniques, bar charts and scatter plots are described here which will be utilized in analysis of metrics results.

#### 1. Bar Charts

A bar chart is a graphical representation of frequencies or magnitudes by rectangles drawn with lengths proportional to the frequencies or magnitudes concerned [44]. A bar chart is similar to histograms but differs in few ways. Usually, bars differ only in length, not in width and, some space is left between the bars to facilitate analysis. When a graphical comparison of two or more qualitative distributions with the same class is desired, a combined bar chart can be used.

**Example:** In a study of two Java projects, each consisting of 71 and 58 classes respectively, number of methods in each class were counted. Figure 4.3 gives two bar charts representing this data.

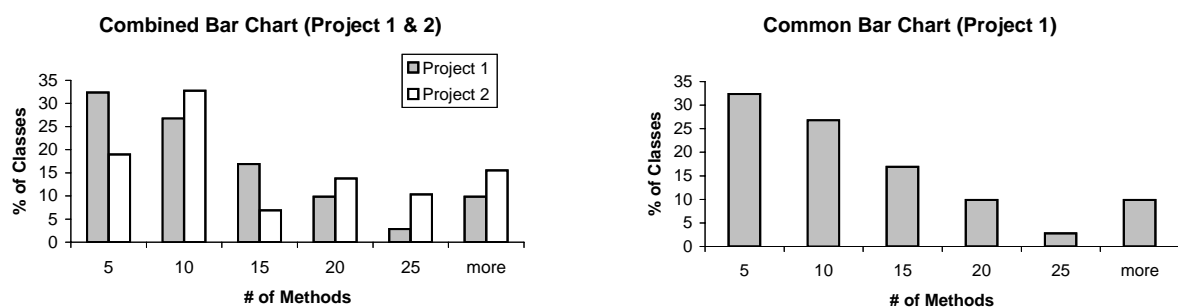


Figure 4.3: Example Bar Charts

#### 2. Scatter Plot

A simple graphic display that is very helpful in studying the relationship between two quantitative variables is a scatter plot [45]. In a scatter plot, the observations of the two

variables for each element in the data set are plotted in a two-dimensional graph. The pattern of the points indicates whether a relationship exists between the two variables and, if so, the nature or type of the relationship.

**Example:** In a study of thirty Java applications, number of classes and methods in each of the applications were counted. Figure 4.4 shows a scatter plot drawn between number of classes and number of methods for those thirty applications. This plot suggests a somewhat linear relationship between the two quantities.

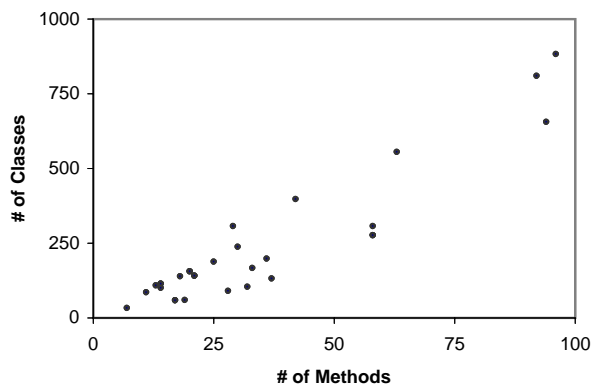


Figure 4.4: An Example Scatter Plot

### 4.2.2 Numerical Statistical Methods

This section contains a description of a few relevant numerical statistics as explained below.

#### 1. Mean

Arithmetic mean, or simply a mean (generally called average) is a measure of central location of a set of data elements. The mean of  $n$  numbers is their sum divided by  $n$  [46].

**Example:** A student achieved following grades in five examinations:

1.3, 1.7, 2.0, 3.3, 1.7

$$\text{mean} = \frac{1.3 + 1.7 + 2.0 + 3.3 + 1.7}{5} = \frac{10.0}{5} = 2.0$$

#### 2. Median

To avoid the possibility of being misled by very small or very large values, we sometimes describe the middle or center of a set of data with another statistical measure, that is median. The median of a set of scores is the middle value when the scores are arranged in order of increasing (or decreasing) magnitude [47]. If the number of elements is odd,

median is the value of middle item. If the number of elements is even then median is the mean of two middle items.

**Example 1:** In a cricket match, five players made these scores:

45, 34, 4, 19, 23

First, arrange the scores in increasing order:

4, 19, 23, 34, 45

$$\text{median} = 23$$

**Example 2:** In another cricket match, six player made these scores:

39, 34, 14, 16, 22, 50

First, arrange the scores in increasing order:

14, 16, 22, 34, 39, 50

$$\text{median} = \frac{22 + 34}{2} = \frac{56}{2} = 28$$

### 3. Mode

The mode of a data set is the score that occurs most frequently [47]. When the scores occur with the same greatest frequency, each one is a mode and the data set is *bimodal*. When more than two scores occur with the same greatest frequency, each is a mode and data set is said to be *multimodal*. When no score is repeated, we say that there is no mode.

**Example:** Consider the following three data sets:

a) 6, 4, 15, 8, 4, 4, 12

b) 5, 10, 5, 2, 6, 4, 9, 10

c) 10, 6, 5, 9, 14, 3

a) Mode is 4 which occurs three times.

b) Both 5 and 10 are modes. The data set is bimodal.

c) There is no mode because no number is repeated.

### 4. Standard Deviation

It is the most widely used measure of dispersion of a frequency distribution introduced by K. Pearson [44]. Standard deviation of a set of sample scores is a measure of variation of scores about the mean [47]. Sample standard deviation (calculated for a subset-population *sample*) is denoted by  $s$  and is calculated by formula 4.1:

$$s = \sqrt{\frac{\sum(x - \bar{x})^2}{n - 1}} \quad (4.1)$$

where:

$x$  is the value of item,

$\bar{x}$  is the sample arithmetic mean, and

$n$  is the sample size

A slightly different formula is used to calculate standard deviation  $\sigma$  of a *population*. This formula is given in equation 4.2

$$\sigma = \sqrt{\frac{\sum(x - \mu)^2}{N}} \quad (4.2)$$

where:

$x$  is the value of item,

$\mu$  is the population arithmetic mean, and

$N$  is the population size

### 5. Skewness

Skewness summarizes the extent to which the observations are symmetrically distributed [45]. The skewness of a data set is readily studied if the data are presented as a frequency distribution. Positive skewness indicates a right-tailed distribution while negative skewness indicates a left-tailed distribution. Skewness for a normal distribution is zero. Figure 4.5 gives examples of frequency distributions for different types of skewness.

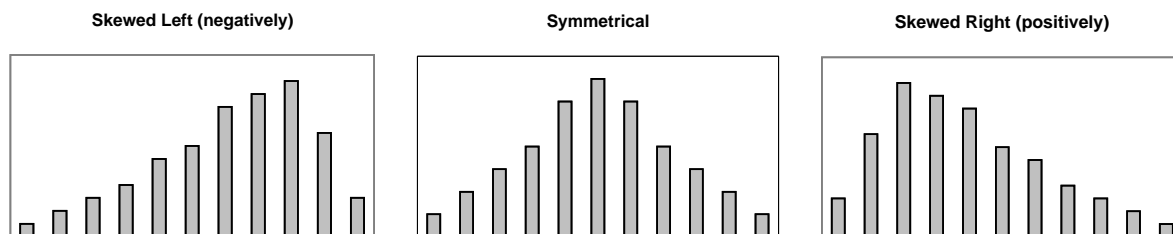


Figure 4.5: Examples of Skewness

### 6. Kurtosis

Kurtosis is a term used to describe the extent to which a unimodal frequency curve is 'peaked'; that is to say, the extent of the relative steepness of ascent in the neighborhood of the mode [44]. Karl Pearson defined it as a normalized form of the fourth central moment of a distribution given by the equation 4.3.

$$\beta_2 = \frac{\mu_4}{\mu_2^2} \quad (4.3)$$

where  $\mu_i$  denotes the  $i$ th central moment (and in particular,  $\mu_2$  is the variance). Positive kurtosis indicates a relatively peaked distribution while negative kurtosis indicates a relatively flat distribution. It is doubtful, however, whether any single ratio can adequately measure the quality of 'peakedness' [44]. Figure 4.6 gives example frequency distributions for positive and negative kurtosis.

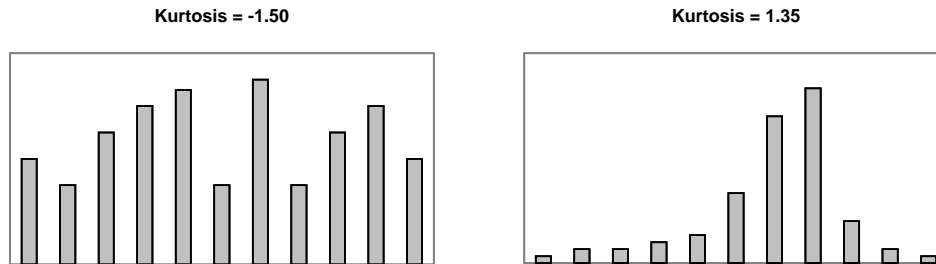


Figure 4.6: Examples of Kurtosis

#### 7. Jarque-Bera

The Jarque-Bera (JB) is a statistic that shows if a sample could have been drawn from a normal distribution [48]. It is based on the sample kurtosis and skewness. A JB statistic of 0 indicates that the distribution has a skewness of 0 and a kurtosis of 3, and is therefore judged to come from a normal distribution. Skewness values other than 0 and kurtosis values other than 3 lead to increasingly large JB values [48]. This statistic is calculated by equation 4.4.

$$JB = \frac{n}{6} \left( S^2 + \frac{(K - 3)^2}{4} \right) \quad (4.4)$$

where  $S$  represents the skewness,  $K$  is the kurtosis, and  $n$  is the number of observations.

JB statistic values can be used to create a test to tell if one can be 95% confident that a sample comes from a normally-distributed population.

#### 8. Kolmogorov-Smirnov Test

This test is used to decide if a sample comes from a population with a specific distribution. It uses cumulative distribution function ( $CDF$ ). If  $F(x)$  is the population distribution function and  $F_n(x)$  the empirical cumulative distribution function of the sample, then the test makes use of the statistic  $d = \max\{|F(x) - F_n(x)|\}$  [44]. A one-sample Kolmogorov-Smirnov test is used to compare an empirical cumulative distribution function ( $CDF$ ) with a theoretical  $CDF$ . Its main applications are for testing goodness-of-fit with the normal distributions. This test has the advantage of making no assumption about the distribution of the data.

#### 9. Confidence Interval

It is an inferential statistic employed for estimating the value of a population parameter.

A *confidence interval* is a range of values that is likely to contain the true value of the population parameter [47]. It is usually associated with a level or degree of confidence. The degree of confidence is the probability (often expressed in percentage value) that the confidence interval contains the true value of the population parameter [47]. If  $U$  and  $V$  are observable random variables whose probability distribution depends upon some unobservable parameter  $\theta$ , and the relation

$$P(U < \theta < V) = x$$

where  $x$  is a number between 0 and 1, then the random interval  $(U, V)$  is a  $100 \cdot x\%$  confidence interval for  $\theta$  and  $x$  represents the confidence level.

#### 10. Pearson Product-Moment Correlation Coefficient

Suppose we have performed  $n$  measurements  $(x_1, y_1), \dots, (x_n, y_n)$  on a pair of two random variables  $x$  and  $y$ . The Pearson product-moment correlation coefficient (or generally called linear correlation coefficient) measures the strength of the linear relationship between the pairs  $x$  and  $y$  in a sample [47] (see also [46]). It is denoted by  $r$  and the formula is given in equation 4.5:

$$r = \frac{S_{xy}}{\sqrt{S_{xx} \cdot S_{yy}}} \quad (4.5)$$

where:

$$S_{xx} = \sum x^2 - \frac{1}{n} (\sum x)^2$$

$$S_{yy} = \sum y^2 - \frac{1}{n} (\sum y)^2$$

$$S_{xy} = \sum xy - \frac{1}{n} (\sum x) (\sum y)$$

The value of  $r$  ranges between -1 and 1, while the absolute value of  $r$  gives the strength of relationship and the sign indicates the direction of the linear relationship. A positive value indicates a direct linear relationship, while a negative value indicates an inverse relationship. The Pearson product-moment correlation coefficient is based on following assumptions [49]:

- The sample of  $n$  subjects for which the value  $r$  is computed is randomly selected from the population it represents,
- The level of measurement upon which each of the variables is based is interval or ratio, and
- The two variables have bivariate normal distribution.

It is important to note that correlation does not imply causation [49]. This is the case, since extraneous variables which have not been taken into account can be responsible for the observed correlation between the two variables.

#### 11. Spearman Rank Correlation

Sometimes, the stringent assumptions made by Pearson product moment correlation coefficient may not be applicable to our data set and we have to choose some other measure of correlation. Spearman rank correlation is a nonparametric, distribution-free rank statistic which measures the strength of the association between two variables [49].

To calculate the rank-correlation coefficient for a given set of paired data, we first rank the  $x$ 's among themselves from low to high or high to low; then we rank  $y$ 's in the same way, find the sum of the squares of the differences,  $d$ , between the ranks of the  $x$ 's and the  $y$ 's, and substitute into the formula [46] (given in equation 4.6),

$$r_s = 1 - \frac{6(\sum d^2)}{n(n^2 - 1)} \quad (4.6)$$

where  $n$  is the number of pairs of  $x$ 's and  $y$ 's. When there are ties in rank, we proceed as before and assign to each of the tied observations the mean of the ranks which they jointly occupy.

It is asserted again that, measures of correlation are not inferential statistical tests, but are, instead, descriptive statistical measures which represent the degree of relationship between two more more variables. Upon computing a measure of correlation, it is common practice to employ one ore more inferential statistical tests in order to evaluate one or more hypotheses concerning the correlation coefficient [49].

## 4.3 Measurement Evaluation

### 4.3.1 Size Metrics Results

Some size metrics for these measured technologies are given in table 4.1. Some more than 10,000 Java class files have been analyzed in this study. The measurement and analysis web application will make available even more data in future as new user applications are submitted to it for evaluation.

### 4.3.2 Chidamber & Kemerer Metrics Results

This section discusses the results of Chidamber & Kemerer metrics evaluations for the aforementioned libraries/applications. Empirical software metrics results usually contain a narrow range of most commonly used values. Here, we are going to explore those commonly observed values forming the basis of our experience-based metrics heuristics. A important

Table 4.1: Size Metrics for Measured Technologies

	# Classes	# Methods	Loc/Class	Methods/Class
J2SE 1.5.0	5221	55846	55	10
J2EE 1.4	746	5134	24	6
J2EE 1.3.1	1062	7427	17	6
J2EE 1.2.1	1127	8328	19	7
JWSDP 1.5	2613	24827	32	9
Tomcat 5.5.9	682	9574	114	14
Tomcat 4.1.31	451	4873	66	10
JavaCard 2.2.1	570	4405	50	7
J2ME	101	551	3	5
Dom4j <sup>a</sup>	337	5340	54	15
Apache Misc. <sup>b</sup>	608	5480	42	9
JFreeChart 1.0	469	6048	74	12
Cewolf 0.10.3	106	480	16	4
Quilt 0.6-a-5	69	590	47	8
HtmlUnit 1.6	210	1881	29	8

<sup>a</sup>versions 1.6.1 & 1.5.2

<sup>b</sup>Bcel 5.1, Cactus 1.7, Lucene 1.4.3

step in statistically analyzing a given data set is to determine the type of frequency distribution shown by the sample. This is important to know as certain statistical tests and measures are only valid for one type of distribution or the other. To analyze the metrics results, following null hypothesis is formed:

H0- The data is normally distributed.

H1- The data is not normally distributed.

A statistical and graphical observation of the metrics results follows to test this hypothesis.

### Descriptive Statistics

While describing an empirical data collection experiment, a number of statistical measures are used to provide an initial look over the observed data. Few numerical statistics discussed in section 4.2.2 are being employed here to discover certain characteristics of the metrics results.

Table 4.2 contains some initial descriptive statistics for C&K metrics evaluations. This table shows a very high maximum value and standard deviation for some of the C&K metrics. As the sample size is very large, with 10,213 Java classes from different libraries, it has been observed that only about 4% of these classes contain very high outlying values. These outliers are the cause of such anomalous initial statistics.

Table 4.2: Initial Descriptive Statistics: C&amp;K Metrics for All Libraries

	<b>DIT</b>	<b>NOC</b>	<b>WMC</b>	<b>CBO</b>	<b>RFC</b>	<b>LCOM</b>
Minimum	0	0	0	0	0	0
Maximum	6	50	287	196	494	36969
Std. Dev	1.08	2.08	15.17	8.00	28.54	851.12

Tables 4.3 (for J2SE 1.5.0 alone) and 4.4 (cumulative for complete metrics result set) contain some descriptive statistics, after removal of the outliers discussed above.

Table 4.3: Descriptive Statistics: C&amp;K Metrics for J2SE 1.5.0

	<b>DIT</b>	<b>NOC</b>	<b>WMC</b>	<b>CBO</b>	<b>RFC</b>	<b>LCOM</b>
Minimum	0.00	0.00	0.00	0.00	0.00	0.00
Maximum	3.00	2.00	41.00	10.00	69.00	544.00
Mean	0.52	0.13	9.18	1.26	14.68	43.38
Median	0.00	0.00	2.00	0.00	1.00	0.00
Mode	0.00	0.00	6.00	0.00	9.00	4.00
Std. Dev.	0.76	0.41	8.44	2.03	16.03	89.60
Kurtosis	0.93	10.24	1.83	3.32	1.25	9.33
Skewness	1.31	3.27	1.49	1.93	1.39	2.99

A cursory glance at the tables 4.3 and 4.4 shows that both these tables exhibit similar trends for the metrics values. The cumulative results and those for J2SE 1.5.0 have almost similar values of mean and standard deviation for various C&K metrics. High values of skewness and kurtosis indicate that data does not tend to be normally distributed. It is reminded here that skewness summarizes the extent to which the observations are symmetrically distributed while kurtosis is used to describe the extent to which a unimodal frequency curve is 'peaked' (see explanation of skewness and kurtosis in section 4.2.2). These two statistics predict skewed and 'peaked' distributions for all of the C&K metrics. We need further graphical and statistical analysis to confirm this initial assumption of a skewed distribution.

### Bar Charts and Distribution Analysis

Figure 4.7 shows bar charts for Chidamber & Kemerer metrics. It shows cumulative results for all metrics as well as for J2SE 1.5.0 alone. Usually, histograms are used for distribution analysis but these bar charts are similar to histograms because equal class interval has been used for drawing each of the chart for various C&K metrics. Regarding the shape of distribution curve, these charts support our initial assumption that metrics data is not normally distributed

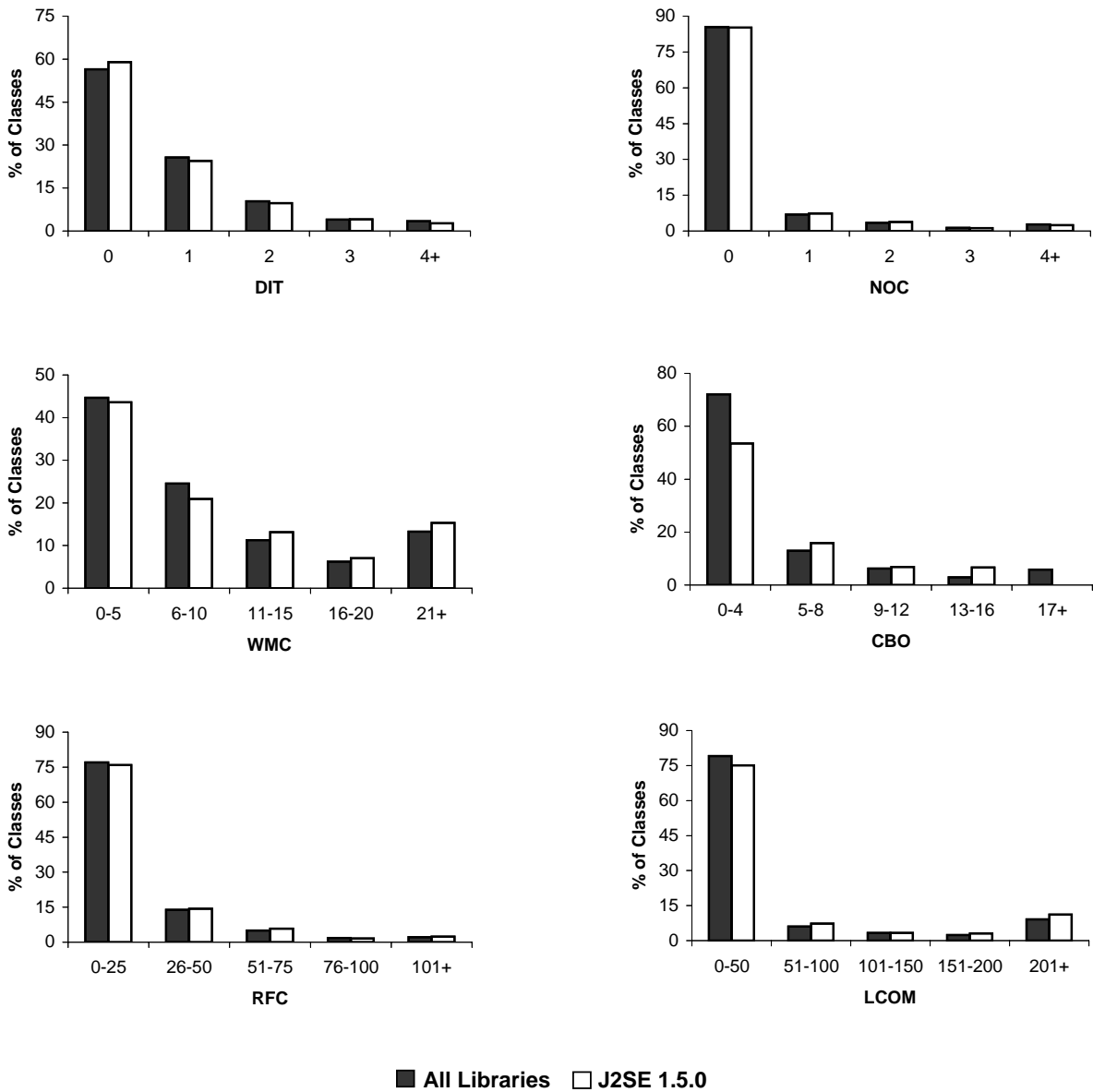


Figure 4.7: Bar Charts for C&K Metrics

Table 4.4: Descriptive Statistics: C&amp;K Metrics for All Libraries

	<b>DIT</b>	<b>NOC</b>	<b>WMC</b>	<b>CBO</b>	<b>RFC</b>	<b>LCOM</b>
Minimum	0.00	0.00	0.00	0.00	0.00	0.00
Maximum	3.00	3.00	40.00	20.00	76.00	556.00
Mean	0.59	0.15	8.69	3.00	14.78	37.51
Median	0.00	0.00	6.00	1.00	8.00	4.00
Mode	0.00	0.00	2.00	0.00	1.00	0.00
Std. Dev.	0.82	0.45	7.90	4.22	16.35	82.28
Kurtosis	0.78	9.70	2.34	2.69	1.95	12.08
Skewness	1.27	3.19	1.62	1.76	1.55	3.35

and rather resembles some kind of positively skewed distribution. We further need to confirm the type of distribution with some robust statistical tests.

Jarque-Bera and Kolmogorov-Smirnov test (section 4.2.2 contains more details) for one sample have been performed at 5% significance level for all six C&K metrics evaluated in this work. The tests have been executed using the functions provided in Matlab 6.5 tool's statistics toolbox. Matlab<sup>1</sup> is both a numerical computing environment as well as a programming language. The aforementioned Matlab functions for Jarque-Bera and Kolmogorov-Smirnov test return a value of 1 if the null hypothesis of normal distribution can be rejected while 0 otherwise. The results are provided in table 4.5.

Table 4.5: Distribution Test Results for C&amp;K Metrics

	<b>Jarque-Bera Test</b>	<b>Kolmogorov-Smirnov Test</b>
DIT	1	1
NOC	1	1
WMC	1	1
CBO	1	1
RFC	1	1
LCOM	1	1

The results indicate a rejection of null hypothesis of normal distribution described in the beginning of section 4.3.2 . Thus the metrics distributions exhibited in this study are not normal. In fact, as can be seen from the figure 4.7 as well, the metrics show some kind of skewed distribution. Skewed distributions are common in empirical software engineering [9]. This type of distribution sometimes complicates statistical analysis as many statistical tests are based on assumption of normal distribution. However, in this case, a skewed distribution is not much a

<sup>1</sup><http://www.mathworks.com/products/matlab/>

concern because we are interested only in studying most commonly observed metrics value patterns.

Seen from another view point, these bar charts show that metrics values for J2SE 1.5.0 alone and those combined over all measured libraries manifest more or less similar value ranges. Another interesting reflection is that a high percentage of classes tend to be within a particular small range of value of each of these metrics. For example, DIT is close to 0 to 1, NOC to 0, WMC 0 to 10, CBO 0-8, RFC 0 to 50 and LCOM 0 to 50 for three fourth of the total classes. Confidence interval statistic is being used next to statistically draw more accurate estimate of these metrics ranges.

### Confidence Intervals

Confidence intervals are being calculated here for all C&K metrics. Although these metrics distributions are not normal and metrics values show a relatively high standard deviation, yet a review of confidence intervals can provide useful information about most probable metrics values.

Confidence intervals at 95% significance level for C&K metrics (based on results from all libraries) are given in table 4.6.

Table 4.6: 95% Confidence Interval:Mean of C&K Metrics (all Libraries)

	<u>Lower Limit</u>	<u>Upper Limit</u>
DIT	0.57	0.61
NOC	0.14	0.16
WMC	8.54	8.85
CBO	2.92	3.09
RFC	14.45	15.10
LCOM	35.88	39.14

### Metrics Heuristics

Combining the information in figure 4.7 and that in table 4.6, following metrics heuristics can be derived. These are not any cut-off thresholds for the concerned metrics. Rather, these values have widely been practised as shown in results and form a likely basis for other development activities based on Java technologies.

Depth of inheritance tree (DIT) is observed to be quite small for most of the classes. It is zero in 56% of cases and less than or equal to 1 for 80% of the studied classes. DIT values of up to 3 are also observed in some applications. Inheritance depth has been found to be correlated to maintainability [27, 50]. Deeper classes tend to increase complexity and are thus avoided.

Number of children (NOC) also shows similar distribution as compared to DIT, rather a higher percentage of classes (85%) tend to have a zero NOC value. It is also common to see NOC up to 3. A higher number of children means a higher reuse but on the other hand also shows an improper class abstraction. NOC has been validated to be correlated to fault proneness [31]. Classes having higher NOC value should be given more attention and thorough testing.

Weighted method per class (WMC) becomes a system size measure if we assume complexity of each method as unity. It is observed to be less than or equal to 5 for 44% of the classes and up to 10 for 68% of the classes. Similar values can be observed in table 4.6 of confidence intervals. While this is not a cut-off value, yet extremely high number of methods should trigger designer's attention for rework. The larger the number of methods in a class, the greater the potential impact on inheriting classes. Consequently, more effort and time are needed for maintenance and testing. WMC is related to fault proneness of a class [33, 31].

Coupling between objects affects change proneness [21] and higher inter-object class coupling calls for rigorous testing [51]. Therefore a smaller CBO is advocated. Value of CBO is up to 3 for 67% of cases and less than or equal to 10 for 88% of the classes in this experiment. RFC is observed to be less than or equal to 15 for 62% of the classes and up to 20 for 70% of the classes. Response for a class is similar kind of coupling measure which focusses on coupling between classes. A higher class coupling reduces its testability and understandability.

LCOM quantifies lack of cohesion between methods inside a class. Its value is up to 25 for 70% of the classes and less than or equal to 50 for 79% of the classes.

### **Size Independence Analysis**

Metrics are used to measure various projects of varying size. If metrics other than the ones specifically designed to measure size also depend on it, no cumulative knowledge will be achieved [35]. Discussing about desirable properties of software metrics, Abreu [3] suggests that non-size metrics should be system size independent. Therefore, an attempt is being made to analyze correlation of C&K metrics with size of the system (using lines of code). Specifically, following null hypothesis about correlation of our metrics with system size is formed.

H0- The metrics are not correlated to system size.

H1- The metrics are correlated to system size.

Before diving into statistical correlation analysis and reaching at some conclusions, one must give attention to pre-conditions for such analysis. Babbie [52] identifies following three criteria for evaluating a cause-and-affect relationship between two sets of variables.

1. The first requirement in a causal relationship between two variables is that the cause precedes the effect in time or as shown clearly in logic.
2. The second requirement in a causal relationship is that the two variables be empirically correlated with one another.

3. The third requirement for a causal relationship is that the observed empirical correlation between two variables is not because of a spurious relationship.

The C&K metrics data set available for this correlation study is very large, that is 10,213 classes. Usually, correlation is computed for a small sample of  $n$  subjects randomly selected from the population it represents [49]. When the correlation coefficient was computed by randomly selecting 10 samples each of size 50 for these C&K metrics, the results were quite different from one sample to another. Statistical techniques exist to effectively evaluate and analyze correlation for a very large population such as this, but such analysis is beyond the scope of this thesis. So we restrict our attention to randomly selecting only one small Java package/application (with fewer than 100 classes) and analyze it for metrics correlations. *java.net* package, which is part of J2SE 1.5.0 set of libraries, has thus been selected for computing correlations.

Scatter plots are being used here to identify a possible relationship of C&K metrics with some size metric. Figure 4.8 gives a glance at relationships between C&K metrics and lines of code (LOC) metric for the *java.net* package. The given plots suggest some linear association of LOC with WMC & RFC. The rest of the metrics plots are quite randomly dispersed and do not suggest any linear or otherwise shape. We need further tests to confirm these observations.

Both Pearson product-moment correlation and Spearman rank correlation are being used here to verify the null hypothesis of correlation. Linear correlation coefficient is a parametric statistic and assumes normal distribution of data under evaluation (see section 4.2.2). As C&K metrics values do not follow normal distribution (as discussed in sections 4.3.2), linear correlation coefficient becomes unsuitable. Spearman rank correlation is a non-parametric test which does not depend on the underlying data distribution.

Each of the evaluated metrics have been tested for correlation against some size metric. Table 4.7 shows linear and rank correlation coefficients calculated between Chidamber & Kemerer metrics and *lines of code* (LOC) based on J2SE 1.5.0's *java.net* package consisting of 60 classes ( $n = 60$ ).

	DIT	NOC	WMC	CBO	RFC	LCOM
Linear Corr. Coef.	0.136	0.078	0.870	0.592	0.930	0.774
Rank Corr. Coef.	0,062	0.084	0.830	0.427	0.908	0.519

Sheskin [49] mentions that upon computing a measure of correlation, it is common practice to employ one or more inferential statistical tests in order to evaluate one or more hypotheses concerning the correlation coefficient. To test the the null hypothesis of correlation, the correlation coefficients given in table 4.7 are compared with those in table of critical values for Pearson correlation coefficient (table A-6 in [47]) and table of critical values for Spearman rank correlation coefficient (table A18 in [49]). If the absolute value of the computed correlation

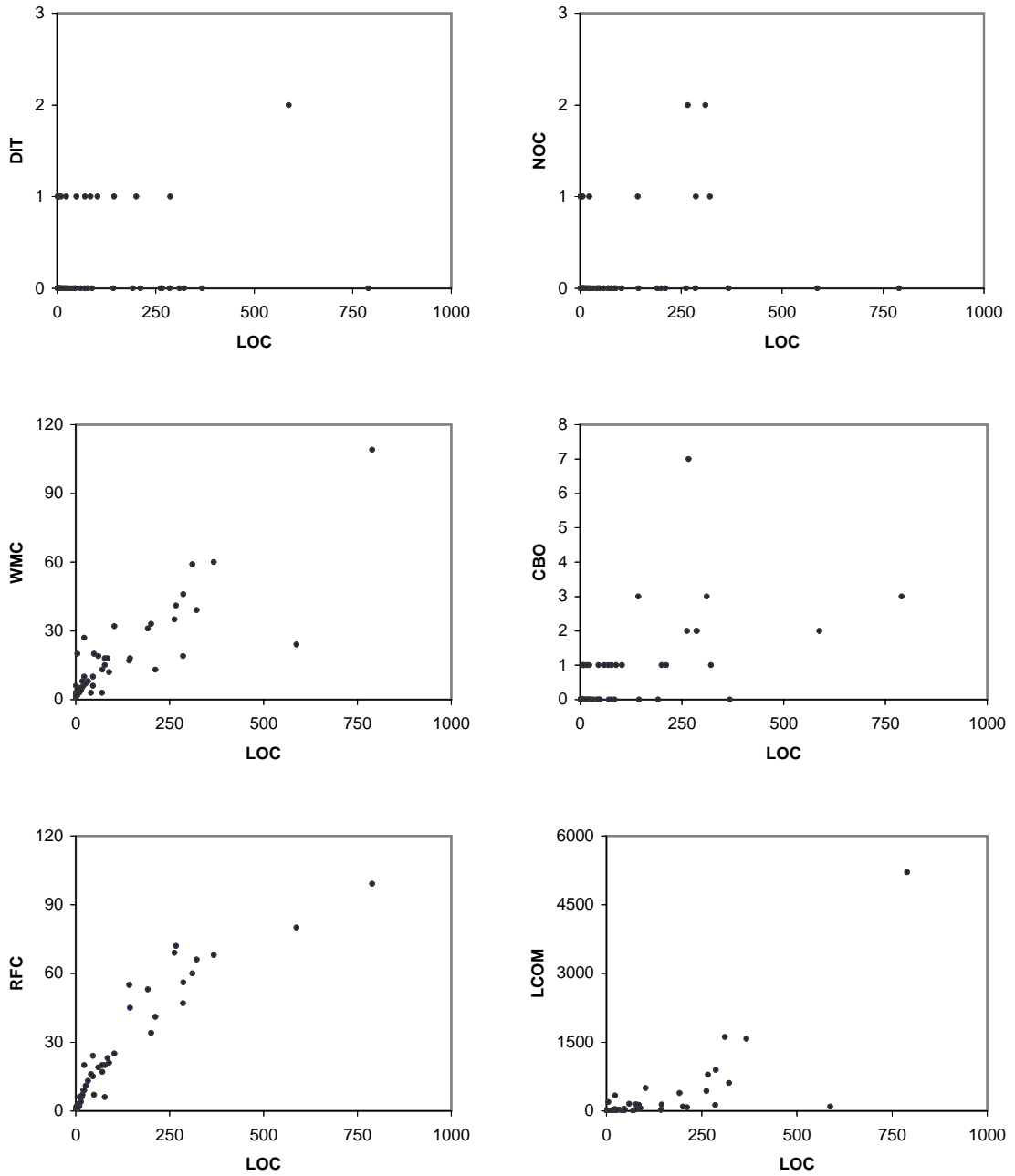


Figure 4.8: Scatter Charts for C&K Metrics

coefficient exceeds the value in the table, we conclude that there is a significant linear correlation. Otherwise, there is not sufficient evidence to support the conclusion of a significant linear correlation [47].

The critical value for linear correlation coefficient at  $n = 60$  and  $\alpha = 0.05$  (significance level) is 0.254 while the critical value for Spearman rank correlation coefficient at  $n = 60$  and  $\alpha = 0.05$  (significance level) is 0.214. Therefore, the null hypothesis of no correlation cannot be rejected for DIT and NOC while it can be rejected for WMC, CBO, RFC and LCOM. Before drawing any conclusion about these metrics correlations, the conceptual relationship of these metrics with LOC count as well as the scatter plots in figure 4.8 have to be considered.

In case of DIT and NOC, there should be no problem in accepting the null hypothesis of correlation,  $H_0$ . DIT and NOC are not conceptually related to lines of code. There may be small classes with few lines of code inside them but designed in a long hierarchical way to give high values of DIT and NOC and vice versa. This is more true about interfaces which do not contain any executable code. The scatter plots of DIT/NOC with LOC also do not suggest any linear shape. Therefore, we conclude that they are not correlated to system size.

Correlation of WMC/RFC with LOC bears similar logic. Comparison of correlation coefficients of WMC and RFC with critical values suggest a rejection of null hypothesis of correlation. Both show strong correlation with LOC and their scatter plots also allude towards a linear shape with few outliers. Methods contain statements and expressions to provide some functionality. A class with larger number of methods (WMC) will eventually have a higher LOC count and hence the relationship. Recall that RFC (response for class) is the number of methods that can be executed in response to a message received by an object of that class. More code inside methods (a higher LOC) performs some functionality by calling methods, thus increasing RFC and showing up a strong correlation.

Although null hypothesis has to be rejected for both CBO and LCOM, but the scatter plots show widely dispersed values without any particular shape. Therefore, the correlation for CBO and LCOM will be regarded as a result of coincidence.

### Metrics Orthogonality

Metrics should capture a unique feature of software design not measured by any other metric. Thus they should be independent of each other, because otherwise they would be redundantly measuring the same property of design.

Figure 4.9 shows scatter plot among selected C&K metrics (those showing high correlation coefficient). WMC drawn against RFC shows a weak linear shape. WMC versus LCOM shows that LCOM remains at low values until a certain value of WMC and beyond it, it increases rapidly. Furthermore, those observations causing a sharp up-slope are few outlier pairs. All the remaining scatter plots are enough dispersed to rule out possibility of a linear relationship.

Tables 4.8 and 4.9 respectively show results of Pearson product-moment correlation coefficient and Spearman rank correlation coefficient among Chidamber & Kemerer metrics based on J2SE 1.5.0's *java.net* package. When these values are compared with those in the tables of critical values for Pearson and Spearman correlation discussed above (0.254 and 0.214), at

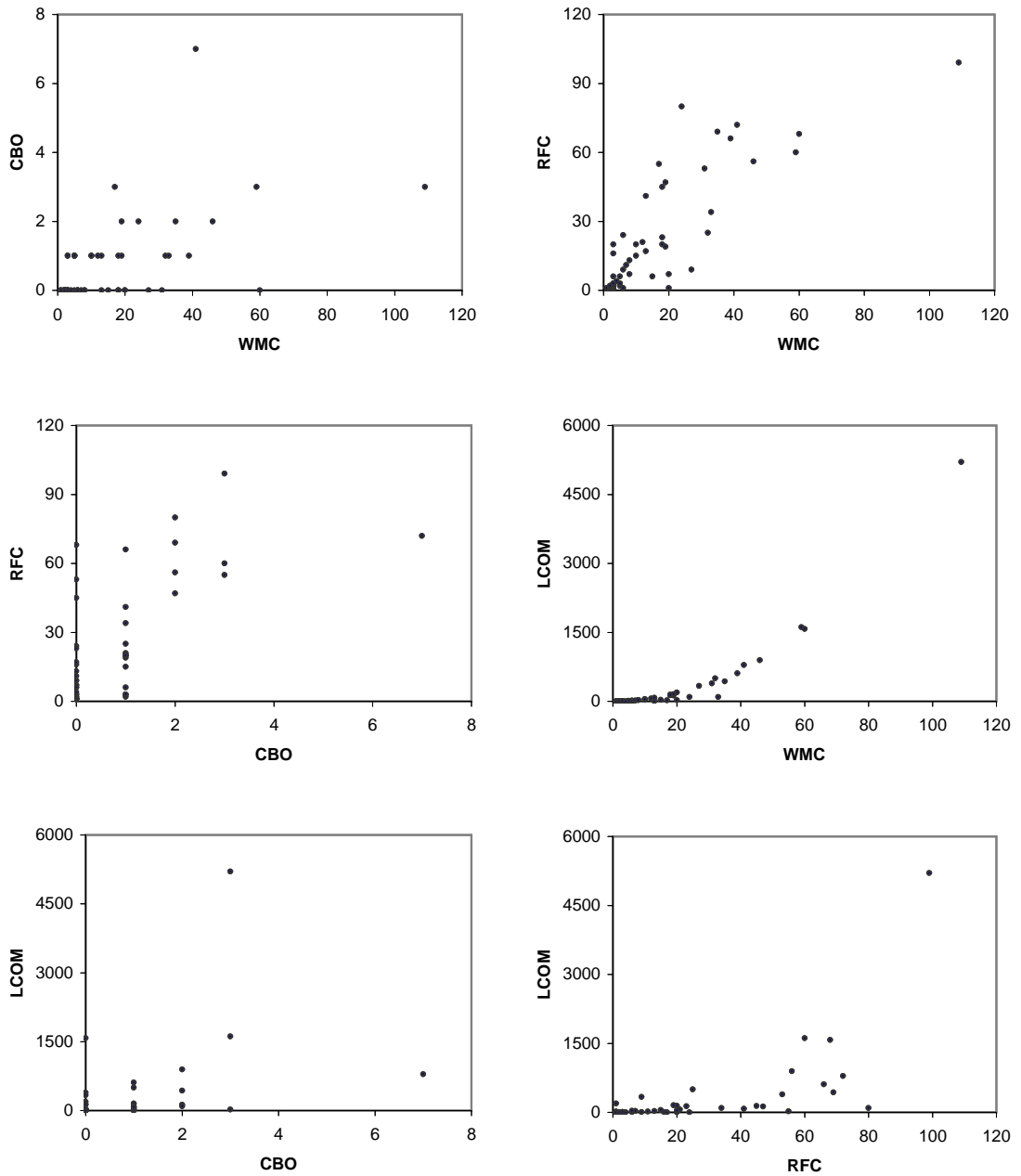


Figure 4.9: Scatter Charts among selected C&K Metrics

least seven metrics pairs show significant correlation. But, when we look at the scatter plots in figure 4.9 only WMC-RFC and WMC-LCOM pairs show some linear shape. So we must reject correlation coefficients for other metrics pairs. RFC is number of possible method calls from an object. With increased functionality provided by more methods, there is potential of increased method calls, that is increased RFC. Recall from definition of LCOM that it is number of dissimilar methods in a class. As we measure more methods and they are also dissimilar, we will eventually observe stronger correlation between WMC and LCOM. This is advocated by high LCOM value in table 4.4.

Table 4.8: Linear Correlation Coefficient among C&K Metrics

	DIT	NOC	WMC	CBO	RFC
NOC	-0.183				
WMC	0.010	0.166			
CBO	-0.057	0.331	0.559		
RFC	0.084	0.140	0.839	0.688	
LCOM	-0.093	0.105	0.890	0.432	0.648

Table 4.9: Rank Correlation Coefficient among C&K Metrics

	DIT	NOC	WMC	CBO	RFC
NOC	0.368				
WMC	0.249	0.405			
CBO	0.291	0.502	0.623		
RFC	0.243	0.333	0.861	0.705	
LCOM	0.209	0.426	0.903	0.615	0.758

### 4.3.3 MOOD Metrics Results

Results of MOOD metrics evaluations follow continuous data representation. We are first interested in finding the type of distribution exhibited by this metrics data set. Similar to discussion for C&K metrics in section 4.3.2, following null hypothesis is formed:

H0- The data is normally distributed.

H1- The data is not normally distributed.

Various statistical techniques will be used in next sections to check this hypothesis and perform an analysis of metrics results.

### Descriptive Statistics

Preliminary statistics on metrics values are presented here (see section 4.2.2 for an explanation of these statistics). Tables 4.10 and 4.11 show an overview of the metrics results for various sets of libraries.

Table 4.10: Descriptive Statistics: MOOD Metrics for J2SE 1.5.0

	<b>MHF</b>	<b>AHF</b>	<b>MIF</b>	<b>AIF</b>	<b>POF</b>
Minimum	89.61	90.70	0.00	0.00	0.00
Maximum	100.00	100.00	79.50	90.62	33.33
Mean	99.19	99.36	24.40	20.29	5.54
Median	99.64	99.86	18.34	10.48	2.75
Mode	100.00	100.00	0.00	0.00	0.00
Std. Dev.	1.56	1.66	21.56	23.39	7.48
Kurtosis	30.83	19.59	-0.01	1.29	3.82
Skewness	-5.14	-4.34	0.93	1.40	1.89

Table 4.11: Descriptive Statistics: MOOD Metrics for all Libraries

	<b>MHF</b>	<b>AHF</b>	<b>MIF</b>	<b>AIF</b>	<b>POF</b>
Minimum	76.19	64.86	0.00	0.00	0.00
Maximum	100.00	100.00	84.21	94.10	50.00
Mean	98.77	98.70	25.35	24.75	4.60
Median	99.56	99.73	20.61	16.67	1.37
Mode	100.00	100.00	0.00	0.00	0.00
Std. Dev.	2.92	3.77	22.37	25.70	7.48
Kurtosis	37.69	47.52	-0.35	-0.10	11.44
Skewness	-5.73	-6.20	0.80	0.96	2.95

Tables 4.10 and 4.11 reveal similar metrics results. MHF, AHF & POF show high skewness while MIF & AIF have relatively low skewness. MHF & AHF are negatively skewed with high kurtosis. This indicates left tailed histograms with high peaks for both these metrics. These values of skewness and kurtosis allusion toward skewed distribution of these metrics values. MIF & AIF both show high standard deviation which shows that they are sparsely dispersed over a wide range of values. After examining initial statistics, a pictorial view of data would be helpful in our analysis.

### Bar Charts and Distribution Analysis

Figure 4.10 shows bar charts for the five MOOD metrics that have been evaluated for different libraries. These diagrams (which are similar to histograms) indicate some type of negatively skewed distribution for MHF and AHF and positively skewed distribution for MIF, AIF and POF. Two statistical normality distribution tests, that is Jarque-Bera test of goodness-of-fit for normality and Kolmogorov-Smirnov test discussed in section 4.2.2, have also been executed for MOOD metrics. As described earlier, these tests have been performed using the functions provided in Matlab<sup>2</sup> 6.5 tool's statistics toolbox. These functions for Jarque-Bera and Kolmogorov-Smirnov test return a value of 1 if the null hypothesis of normal distribution can be rejected while 0 otherwise. The results of these tests are provided in table 4.12.

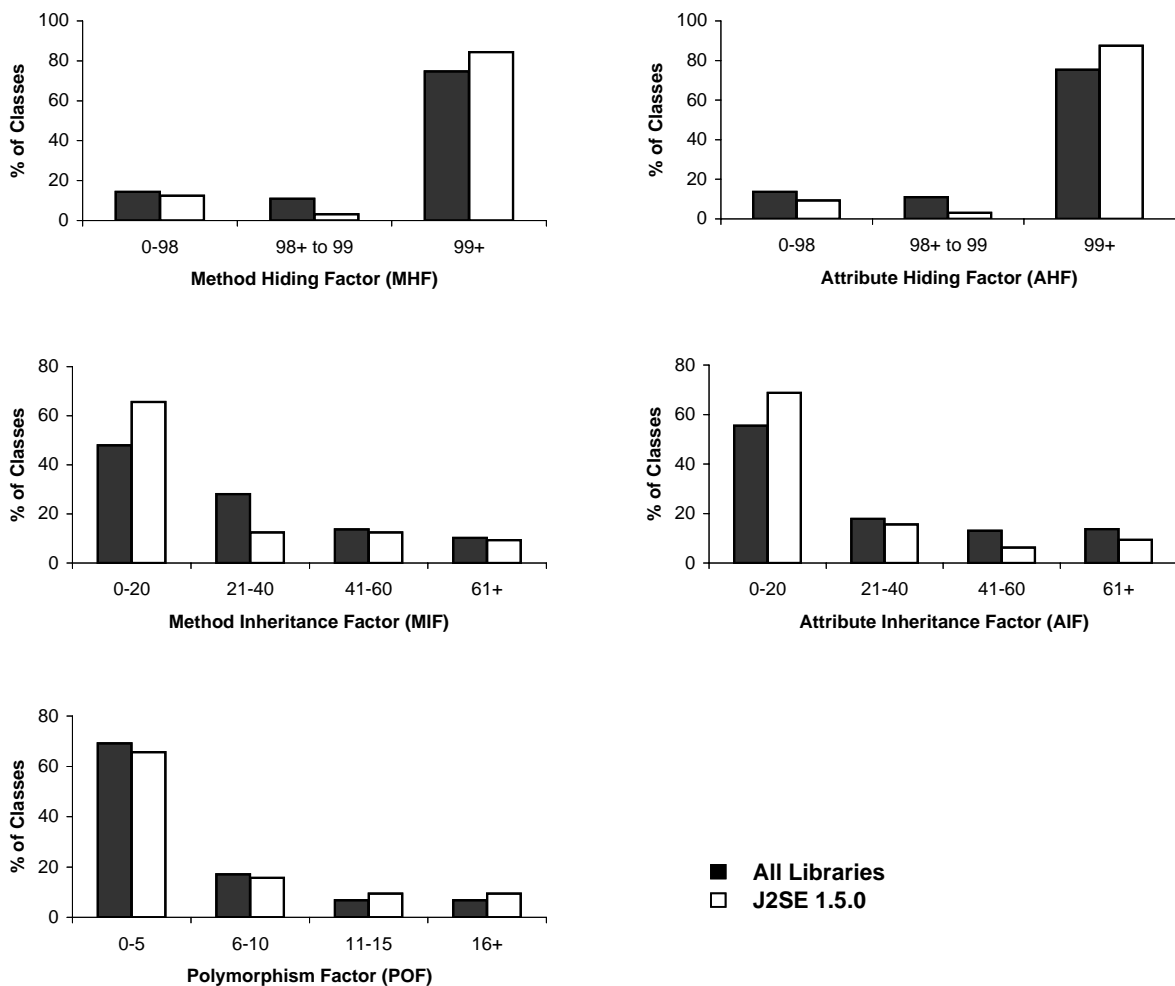


Figure 4.10: Bar Charts for MOOD Metrics

<sup>2</sup><http://www.mathworks.com/products/matlab>

Table 4.12: Distribution Test Results for MOOD Metrics

	Jarque-Bera Test	Kalmogorove-Smirnov Test
MHF	1	1
AHF	1	1
MIF	1	1
AIF	1	1
POF	1	1

As evident from table 4.12 we must reject our null hypothesis of normal distribution formulated in section 4.3.3. Therefore, the metrics values do not show normal distribution. Although the bar charts in figure 4.10 show skewed distributions, we cannot be sure about actual type of distribution until further distribution tests are performed. But, finding such exact type of distribution is not relevant here.

These bar charts, like those for C&K Metrics, show a resemblance between metrics results for *all* evaluated libraries and those provided by J2SE 1.5.0 alone. Furthermore, metrics values show visible range patterns which apply to majority of packages/systems. For example, both MHF and AHF are observed to be close to 99% while MIF and AIF from 0 to 40 and POF from 0 to 10% for most packages. To draw more factual value ranges for these metrics we use confidence interval statistic.

### Confidence Intervals

Although the data distribution is not normal and hence calculations of confidence interval given in table 4.13 can not be much trustful, yet in conjunction with bar charts, these confidence intervals can still reveal some commonly occurring metrics values.

Table 4.13: 95% Confidence Interval:Mean of MOOD Metrics (all Libraries)

	Lower Limit	Upper Limit
MHF	98.30	99.25
AHF	98.09	99.32
MIF	21.73	28.98
AIF	20.58	28.92
POF	3.39	5.81

### Metrics Heuristics

Method hiding factor (MHF) and attribute hiding factor (AHF), part of MOOD metrics set, attempt to capture level of encapsulation implemented by a system. Encapsulation copes with complexity and helps develop a maintainable and reusable software. The libraries in this study show a very high level of encapsulation with both MHF & AHF higher than 98% for most of the observed classes. Ideally, all attributes should be hidden and AHF in this case is close to the optimal value. MHF is also quite high for these set of libraries which seems anomalous. A very high value shows little functionality while low values show improper encapsulation. This very high value of MHF may be due to one of two reasons:

- The Java bindings have been defined incorrectly. For this experiment a top level Java package is being considered as a system in the MOOD metrics sense. Public methods and few others defined with *protected* and no access modifier count towards visibility sum in the numerator of MHF which is far less than denominator (the total number of methods in that package). This seems appropriate until another view of Java binding for this metric becomes available.
- The metric itself is not defined properly and does not fit well to Java programming language concepts. See also a critical analysis of MOOD metrics set [38].

Inheritance is measured by method inheritance factor (MIF) and attribute inheritance factor (AIF). Inheritance on one hand results in reuse but on the other hand may add to complexity of design. MIF is less than 20% for 44% of the classes and less than or equal to 30% for 65% of the classes. Similarly, AIF is below 20% for 52% of classes and less than or equal to 30% for 64% of classes. Similar ranges are predicted by confidence intervals. Values of about 60% to 70% for these metrics were observed for some Eiffel [36] and C++ [35] libraries.

Polymorphism factor (POF) is a measure of polymorphism implemented by a system. Polymorphism should be used very carefully. It allows simplicity by providing dynamic binding but may also complicate tracing control flow within classes. A low value between up to 3% is observed for 55% of classes. 73% of classes have POF value less than or equal to 6%. See [36, 35] for other similar studies.

### Size Independence Analysis

Figure 4.11 contains scatter plots between MOOD metrics and *number of classes* based on 48 packages ( $n = 48$ ) from J2SE 1.5.0. The first plot between *number of methods* and *number of classes* reveals a linear relationship between them. The remaining plots do not form any particular shape and indicate no visible relationship of MOOD metrics with *number of classes* metric. It is also evident from these scatter plots that MHF and AHF lie within a very narrow range of values (see also table 4.11 and figure 4.10), actually close to 98%. These values are not diverse enough to form a basis of a correlation analysis and may be misleading. Therefore, any possible significant correlation coefficient involving MHF or AHF should not be considered trustful.

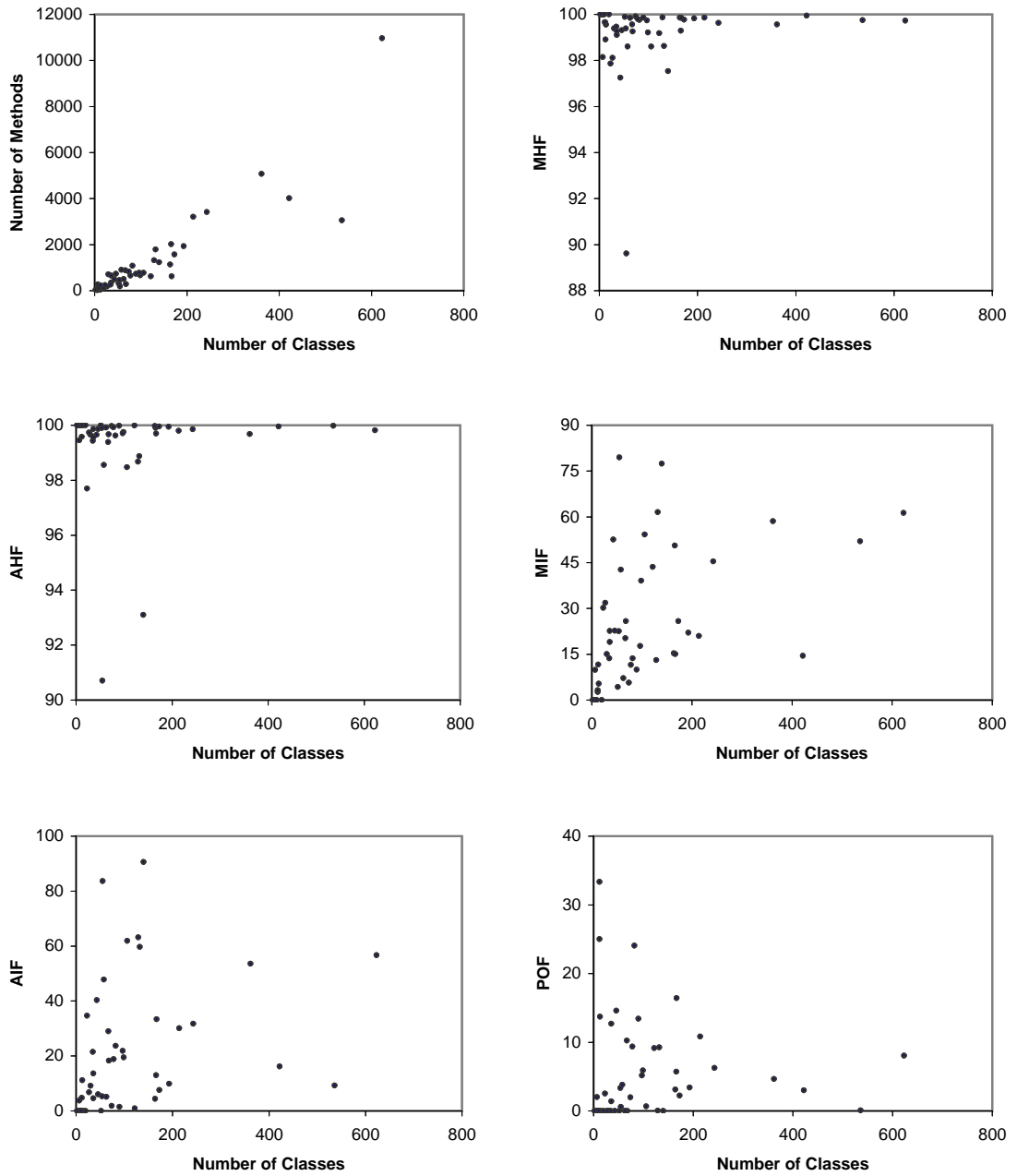


Figure 4.11: Scatter Charts for MOOD Metrics

Table 4.14 gives values of Pearson product-moment correlation coefficient and Spearman rank correlation coefficient for MOOD metrics (see also critical discussion on correlation at the end of section 4.3.2). The critical value for Pearson correlation coefficient for  $n = 48$  and  $\alpha = 0.05$  (significance level) is 0.279 which has been extracted from table A-6 given in [47]. The critical value for Spearman rank correlation at  $\alpha = 0.05$  is 0.279, taken from table A18 given in [49].

Table 4.14: Correlation of MOOD Metrics with Size Metrics

		MHF	AHF	MIF	AIF	POF
Linear Corr. Coef.	#Methods	0.130	0.054	0.474	0.311	-0.021
	#Classes	0.129	0.072	0.442	0.357	0.033
Rank Corr. Coef.	#Methods	-0.049	-0.298	0.599	0.588	0.363
	#Classes	-0.019	-0.229	0.630	0.580	0.350

Pearson correlation coefficient for MIF and AIF shows values higher than the threshold. Absolute value of the rank correlation coefficient for all MOOD metrics, except for MHF, is higher than the threshold. This advocates a statistical correlation but is contradictory to scatter plots for these metrics. In view of the concepts of an object-oriented design, AHF, MIF, AIF or POF should not have a considerable effect on the number of classes in the system. There may be some small applications designed in a way so as to give high values for these metrics and vice versa. Therefore, it is concluded that all these MOOD metrics are not correlated to number of classes or methods in the system.

### Metrics Orthogonality

Figure 4.12 contains scatter charts among selected MOOD metrics. MHF & MIF and AHF & AIF pairs may be inversely correlated as encapsulation can have a reverse effect on inheritance. As has been discussed earlier, MHF and AHF show insufficiently diverse values and may produce biased correlation results. So, somewhat linear shape of scatter plot involving these metrics cannot be considered a result of possible correlation. Apart from them, scatter plot between MIF & AIF shows some kind of linear shape.

Tables 4.15 and 4.16 contain results of two correlation tests (Pearson linear and Spearman rank) performed for MOOD metrics.

Disregarding correlation for any metrics pairs involving MHF/AHF and for those metrics which do not show any linear shape in the scatter plots, discussion will now be focussed on MIF/AIF pair which shows some visible linear shape in scatter plot as well as a considerable value of correlation coefficient (both for Pearson and Spearman). We may wrongfully deduce a correlation between these metrics if we do not take into account one of the requirements for a cause-and-effect relationship discussed at the end of section 4.3.2 which says that a statistical correlation should also be logical in the context involved. In some cases MIF & AIF may be

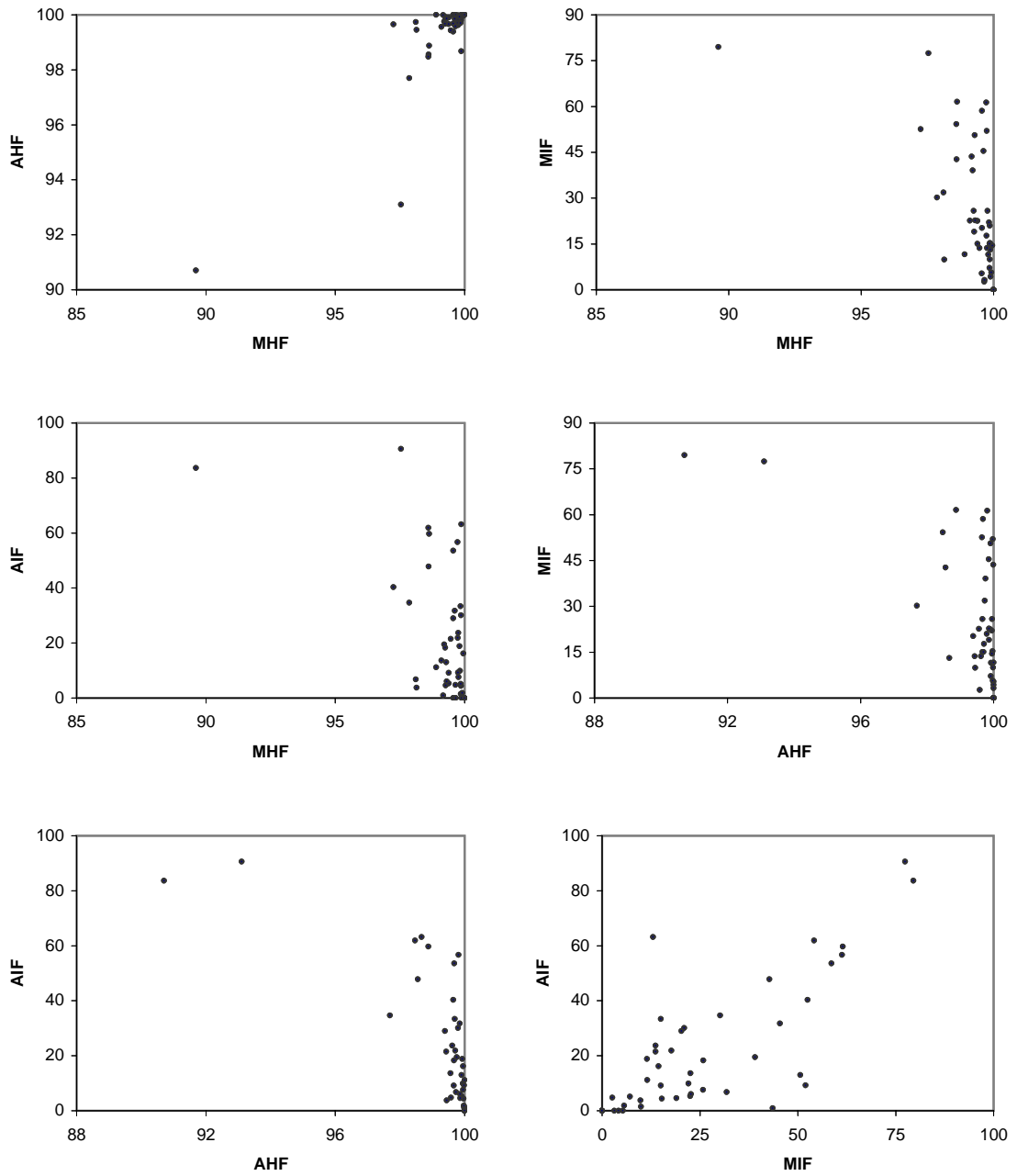


Figure 4.12: Scatter Charts among selected MOOD Metrics

Table 4.15: Linear Correlation Coefficient among MOOD Metrics

	MHF	AHF	MIF	AIF
AHF	0.856			
MIF	-0.571	-0.586		
AIF	-0.545	-0.720	0.772	
POF	0.143	0.156	-0.129	-0.080

Table 4.16: Rank Correlation Coefficient among MOOD Metrics

	MHF	AHF	MIF	AIF
AHF	0.661			
MIF	-0.700	-0.593		
AIF	-0.510	-0.805	0.753	
POF	-0.117	-0.126	0.201	0.262

strongly correlated while in others not. It is because when we subclass a given class, both methods and attributes have equal chance of being inherited, depending upon their access modifier. Usually, most attributes are *private* while methods are defined with different access modifiers depending upon design requirements. If this is the case, there will be a weak correlation among them. But if most of the methods and attributes are defined with a *public* access modifier, we will observe strong correlation between MIF and AIF.

Moreover, MHF/MIF and AHF/AIF should have been inversely correlated. This is because more *private* or hidden methods (causing high MHF) will eventually leave fewer methods to be inherited (resulting low MIF). The same can be said about AHF and AIF. But very narrow range of values for MHF and AHF restricts us from drawing any empirical evidence for such correlations.

## 4.4 Synopsis

Tables 4.17 and 4.18 briefly reiterate the metrics heuristics derived in sections 4.3.2 and 4.3.3 for Chidamber & Kemerer and MOOD metrics, respectively. The metrics values given in both these table reflect the combined information coming from bar charts, confidence interval tables and metrics heuristics discussions in sections 4.3.2 and 4.3.3.

Furthermore, metrics correlation analysis shows that WMC (weighted method per class) and RFC (response for class) from the Chidamber & Kemerer metrics suite are associated with system size. None of the MOOD metrics shows any association with system size.

Table 4.17: C&amp;K Metrics Heuristics

	<b>Metric Value</b>
DIT	1
NOC	0
WMC	9
CBO	4
RFC	20
LCOM	25

Table 4.18: MOOD Metrics Heuristics

	<b>Metric Value (%)</b>
MHF	98.77
AHF	98.70
MIF	29.00
AIF	29.00
POF	6.00



## 5 Conclusions and Future Work

*I think and think for months and years. Ninety-nine times,  
the conclusion is false. The hundredth time I am right.*  
-Albert Einstein

Object-oriented software metrics provide powerful means of assessing, managing and controlling a software process. An effective measurement methodology can transform programming into an engineering activity. Evaluation and analysis of metrics values for standard libraries and other legacy software can expose common development approaches serving as benchmarks for other software projects to be built in future. This thesis presents an approach towards online measurement, analysis, and metrics value storage for Java based libraries and projects. This chapter provides a summary of this work and gives the directions for future work.

### 5.1 Thesis Summary

A metrics-based analysis of an object-oriented design highlights its peculiar features and is very helpful for anticipating effects of design on software quality attributes. This thesis proposes the concept of an online software metrics evaluation and analysis application and provides metrics results by applying Chidamber & Kemerer and MOOD metrics to several standard Java libraries and projects. It also introduces Java bindings for these set of metrics. The analysis of the results reveals helpful cognitions about how object-oriented approach is being implemented by these technologies. In addition to providing common trends in metrics values, this information can be combined with validation studies from other researchers to adapt software design to proven approaches and hence avoid possible expensive maintenance tasks and optimize design for better quality software.

Furthermore, two different correlation techniques (linear and rank) have been employed to study relationship of Chidamber & Kemerer and MOOD metrics with system size metrics. The results indicate no correlation of Chidamber & Kemerer (except *weighted method per class* and *response for a class*) and MOOD metrics with size of the system. Thus, we can assume that both these metric sets provide a unique view of properties of a software design. Results of orthogonality among these metrics themselves are also provided which highlight linear relationships among these metrics.

## 5.2 Future Work

At present, considerable active research is being carried out in the area of software metrics. This thesis work can be extended in few directions as outlined below.

### 5.2.1 Validation Studies using the Metrics Results

Any metrics validation study involves evaluating independent variable (software metrics), the dependent variable (quality attributes such as error proneness/maintainability etc) and then applying statistical techniques to find relationship among both of them. As a result of this thesis, metrics results of a very large set of Java standard libraries are available in a portable format, that is XML files. One can easily validate these metrics against software complexity and/or maintainability by somehow evaluating the later (external quality indicators). These validations, being capable to be based on a variety of several Java class files, can be more reliable than previous studies which were mostly applied to a small data set.

### 5.2.2 Metrics-Based Analysis of other Competing Technologies

This thesis analyzes a specific set of object-oriented metrics for various Java technology libraries. A similar analysis can be performed for other competing technologies such as .NET, C++ etc. Chidamber & Kemerer and MOOD belong to the class of structural and complexity metrics. A metrics based analysis of various programming language libraries can expose structural and design commonalities among them. Thus we can obtain more generalized view of software design heuristics. The analysis data can also reveal comparative inherent complexity within various standard libraries which is likely to be inherited to the software applications using those libraries.

### 5.2.3 Validation of MOOD Metrics

As opposed to Chidamber & Kemerer metrics set, MOOD metrics set has been a focus of only a few empirical validations mainly by the author himself. Only one theoretical validation has been performed by Harrison et. al. [37]. MOOD metrics set is different in being one of few *system* level metrics sets. There is possibility of further theoretical and empirical validation of this metrics set.

### 5.2.4 Object-Oriented Metrics Database

An effective measurement process requires continuous evaluation of different software metrics and integrating them into the software development process. This calls for an efficient storage management of measurement data. A software metrics database can serve this purpose. Dumke et. al. [41] elaborate in detail on the concept of a metrics database. Braungarten [53]

surveys a number of existing storages (spreadsheets, databases, repositories) of software measurement data. Few among them are NASA/SEL<sup>1</sup> Dataset, ISBSG<sup>2</sup> Benchmarking Data CD R8, and NASA MDP Data Repository. These spreadsheets, databases and repositories contain measurement data about various attributes of a wide variety of software projects.

Similar to this concept and as an extension to 5.2.2 discussed above, an object-oriented metrics database can be maintained comprising metrics evaluations based on standard libraries from competing object-oriented languages. The analysis of data can probably help us in understanding why some language libraries are sometimes considered difficult (or easier) to implement than the others. Another possible application is for deriving metrics heuristics, as in this thesis, or for benchmarking purposes.

---

<sup>1</sup><http://sel.gsfc.nasa.gov>

<sup>2</sup><http://www.isbsg.org>



# Bibliography

- [1] Robert. N. Charette. Why software fails. *IEEE Spectrum*, September 2005.
- [2] S. R. Chidamber and C. F. Kemerer. A metrics suite for object oriented design. *IEEE Trans. Softw. Eng.*, 20(6):476–493, 1994.
- [3] F. B. Abreu and R. Carapuça. Object-oriented software engineering: Measuring and controlling the development process. In *Proceedings of the 4th International Conference on Software Quality, ASQC, McLean, VA, USA*, October 1994.
- [4] Reiner R. Dumke and Erik Foltin. Metrics-based evaluation of object-oriented software development methods. In *CSMR*, pages 193–196, 1998.
- [5] Norman E. Fenton. Software measurement: Why a formal approach? In *Formal Aspects of Measurement*, pages 3–27, 1991.
- [6] Reiner R. Dumke. *Software Engineering*. Vieweg Verlagsgesellschaft mbH, Wiesbaden, Germany, 4th revised and extended edition, 2003.
- [7] H. Zuse. *A Framework of Software Measurement*. Walter de Gruyter & Co., Berlin, Germany, 1998.
- [8] Norman E. Fenton. *Software Metrics: A Rigorous and Practical Approach*. International Thomson Computer Press, Boston, MA, USA, 1997.
- [9] Martin Shepperd. *Foundations of software measurement*. Prentice Hall International (UK) Ltd., Hertfordshire, UK, 1995.
- [10] Brian Henderson-Sellers. *Object-oriented Metrics: Measures of Complexity*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996.
- [11] David A. Gustafson and Baba Prasad. Properties of software measures. In *Proceedings of the BCS-FACS Workshop on Formal Aspects of Measurement*, pages 179–193. Springer-Verlag, 1992.
- [12] S. D. Conte, H. E. Dunsmore, and V. Y. Shen. *Software engineering metrics and models*. Benjamin-Cummings Publishing Co., Inc., Redwood City, CA, USA, 1986.

- [13] M Lorenz and J Kidd. *Object-oriented Software Metrics*. Prentice-Hall Object Oriented Series, 1994.
- [14] Lionel C. Briand, P. Devanbu, and W. Melo. An investigation into coupling measures for c++. In *ICSE '97: Proceedings of the 19th international conference on Software engineering*, pages 412–421, New York, NY, USA, 1997. ACM Press.
- [15] Michelle Cartwright and Martin Shepperd. An empirical investigation of an object-oriented software system. *IEEE Trans. Softw. Eng.*, 26(8):786–796, 2000.
- [16] Mei-Huei Tang, Ming-Hung Kao, and Mei-Hwa Chen. An empirical study on object-oriented metrics. In *METRICS '99: Proceedings of the 6th International Symposium on Software Metrics*, page 242, Washington, DC, USA, 1999. IEEE Computer Society.
- [17] Saida Benlarbi and Walcelio L. Melo. Polymorphism measures for early risk prediction. In *ICSE '99: Proceedings of the 21st International Conference on Software engineering*, pages 334–344, Los Alamitos, CA, USA, 1999. IEEE Computer Society Press.
- [18] F. B. Abreu and Walcelio Melo. Evaluating the impact of object-oriented design on software quality. In *METRICS '96: Proceedings of the 3rd International Symposium on Software Metrics*, page 90, Washington, DC, USA, 1996. IEEE Computer Society.
- [19] Lionel Briand, Khaled El Emam, and Sandro Morasca. Theoretical and empirical validation of software product measures. Technical Report ISERN-95-03, 1995.
- [20] Lionel C. Briand and Jürgen Wüst. *Empirical Studies of Quality Models in Object-Oriented Systems*, volume 56. Academic Press., 2002.
- [21] Audun Foyen. Dynamic coupling measurement for object-oriented software. *IEEE Trans. Softw. Eng.*, 30(8):491–506, 2004.
- [22] Khaled El-Emam, S. Benlarbi, N. Goel, and Shesh Rai. A validation of object-oriented metrics. Technical report, NRC/ERB 1063. National Research Council Canada, 1999.
- [23] R. Harrison, S. Counsell, and R. Nithi. Experimental assessment of the effect of inheritance on the maintainability of object-oriented systems. *J. Syst. Softw.*, 52(2-3):173–179, 2000.
- [24] Ramanath Subramanyam and M. S. Krishnan. Empirical analysis of ck metrics for object-oriented design complexity: Implications for software defects. *IEEE Trans. Softw. Eng.*, 29(4):297–310, 2003.
- [25] D. Glasberg, Khaled El-Emam, W. Melo, and N. Madhavji. Validating object-oriented design metrics on a commercial java application. Technical report, National Research Council Canada, 2000.

- 
- [26] Khaled El-Emam, W. Melo, and Javam C. Machado. The prediction of faulty classes using object-oriented design metrics. *J. Syst. Softw.*, 56(1):63–75, 2001.
- [27] Lutz Prechelt, Barbara Unger, Michael Philippsen, and Walter Tichy. A controlled experiment on inheritance depth as a cost factor for code maintenance. *J. Syst. Softw.*, 65(2):115–126, 2003.
- [28] S. R. Chidamber, David P. Darcy, and Chris F. Kemerer. Managerial use of metrics for object-oriented software: An exploratory analysis. *IEEE Trans. Softw. Eng.*, 24(8):629–639, 1998.
- [29] G. Denaro, L. Lavazza, and M. Pezzè. An empirical evaluation of object oriented metrics in industrial setting. In *The 5th CaberNet Plenary Workshop, Porto Santo, Madeira Archipelago, Portugal*, November 2003.
- [30] Lionel C. Briand, J. Daly, V. Porter, and J. Wüst. A comprehensive empirical validation of product measures for object-oriented systems. Technical report, ISERN-98-07, 1998.
- [31] Lionel C. Briand, Jürgen Wüst, John W. Daly, and D. Victor Porter. Exploring the relationships between design measures and software quality in object-oriented systems. *J. Syst. Softw.*, 51(3):245–273, 2000.
- [32] Lionel C. Briand, Jürgen Wüst, and Hakim Lounis. Using coupling measurement for impact analysis in object-oriented systems. In *ICSM '99: Proceedings of the IEEE International Conference on Software Maintenance*, page 475, Washington, DC, USA, 1999. IEEE Computer Society.
- [33] Victor R. Basili, Lionel C. Briand, and Walcelio L. Melo. A validation of object-oriented design metrics as quality indicators. *IEEE Trans. Softw. Eng.*, 22(10):751–761, 1996.
- [34] S. Benlarbi, Khaled El-Emam, N. Goel, and Shesh Rai. Thresholds for object-oriented measures. In *ISSRE '00: Proceedings of the 11th International Symposium on Software Reliability Engineering (ISSRE'00)*, page 24, Washington, DC, USA, 2000. IEEE Computer Society.
- [35] F. B. Abreu, Miguel Coulão, and Rita Esteves. Toward the design quality evaluation of object-oriented software systems. In *Proceedings of the 5th International Conference on Software Quality, Austin, Texas, USA*, October 1995.
- [36] F. B. Abreu, R. Esteves, and M. Goulao. The design of eiffel programs: Quantitative evaluation using the mood metrics. In *Proceedings of TOOLS'96, Santa Barbara, CA, USA*, July 1996.
- [37] Rachel Harrison, Steve J. Counsell, and Reuben V. Nithi. An evaluation of the mood set of object-oriented software metrics. *IEEE Trans. Softw. Eng.*, 24(6):491–496, 1998.

- [38] Tobias Mayer and Tracy Hall. Measuring oo systems: A critical analysis of the mood metrics. In *TOOLS '99: Proceedings of the Technology of Object-Oriented Languages and Systems*, page 108, Washington, DC, USA, 1999. IEEE Computer Society.
- [39] Quansheng Xiao. Object-oriented metrics analysis on java software projects. Department of Information and Software Engineering, George Mason University, Fairfax, VA, USA, 2001.
- [40] Reiner R. Dumke and H. Grigoleit. Efficiency of came tools in software quality assurance. *Software Quality Journal*, 6:157–169, 1997.
- [41] Christof Ebert, Reiner Dumke, Manfred Bundschuh, and Andreas Schmietendorf. *Best Practices in Software Measurement*. SpringerVerlag, 2004.
- [42] Craig Larman. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2001.
- [43] Nick Kassem, Nicholas Kassem, and Enterprise Team. *Designing Enterprise Applications: Java 2 Platform*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2000.
- [44] Yadolah Dodge. *The Oxford Dictionary of Statistical Terms*. Oxford University Press, Inc., 2003.
- [45] J. Neter, W. Wasserman, and G. A. Whitmore. *Applied Statistics*. Simon and Schuster, Inc., fourth edition, 1993.
- [46] John E. Freund and Gary A. Simon. *Modern Elementary Statistics*. Ninth edition, 1988.
- [47] Mario F. Triola. *Elementary Statistics*. Addison Wesley Longman, Inc., seventh edition, 1998.
- [48] G. G. Judge, R. C. Hill, W. E. Griffiths, H. Lutkepohl, and T.-C. Lee. *Introduction to the Theory and Practice of Econometrics*. Wiley, 1988.
- [49] David J. Sheskin. *Handbook of Parametric and Nonparametric Statistical Procedures*. CRC Press, Inc., 1997.
- [50] Barbara Unger and Lutz Prechelt. The impact of inheritance depth on maintenance tasks: Detailed description and evaluation of two experiment replications. Technical report, Fakultät für Informatik, Universität Karlsruhe, Germany, July 1998.
- [51] G. Alkadi and D.L. Carver. Application of metrics to object-oriented designs,. In *Proceedings of IEEE Aerospace Conference*, volume 4, pages 159–163, March 1998.
- [52] E. Babbie. *The Practice of Social Research*. Wadsworth Publishing Co., 4th edition.

- [53] R. Braungarten, M. Kunz, and R. Dumke. An approach to classify software measurement storage facilities. Technical report, University of Magdeburg, Department of Computer Science, Magdeburg, Germany, January 2005.



# A Implementation

- Name of Software  
Object-Oriented Measurement of Java Technologies (OOMJ)
- Intended Users  
Software Architects/Designers/Developers
- Programming Language  
Java
- Libraries Used  
The prototype software was written in Java and uses the following libraries:
  - Java 2 Platform Standard Edition 5.0 ([www.java.sun.com/j2se](http://www.java.sun.com/j2se)),
  - CKJM 1.1 Chidamber and Kemerer Java Metrics ([www.spinellis.gr/sw/ckjm](http://www.spinellis.gr/sw/ckjm)),
  - DOM4J 1.6 XML Framework for Java ([www.dom4j.org](http://www.dom4j.org)),
  - BCEL 5.1 Byte Code Engineering Library ([jakarta.apache.org/bcel](http://jakarta.apache.org/bcel)),
  - CEWOLF 0.10.3 Chart Enabling Web Object Framework ([cewolf.sourceforge.net/](http://cewolf.sourceforge.net/)),
- Development Environment  
NetBeans IDE 4.1
- Development System Specification
  - Intel Pentium 4
  - CPU 2.26 GHz
  - 256 MB RAM
  - Windows XP OS
- Test Web Server  
Apache Tomcat 5.5.9

- Test Web Browsers  
Netscape 8.0.3.3, Mozilla Firefox 1.0.4, Internet Explorer 6.0
- Coding Standard  
Coding standard followed was one described by Sun Microsystems in document titled *Code Conventions for the Java Programming Language*.

# Declaration

I herewith declare that I have completed this work by myself and only with the help of the stated references.

Magdeburg, October 12, 2005

Ayaz Farooq

