On the Use of

Software Metrics

Rohit Kumar

Department of Computer Science and Engineering
National Institute of Technology Rourkela
Rourkela-769 008, Odisha, India
May 2014

On the Use of

Software Metrics

Thesis submitted in partial fulfillment of the requirements for the degree of

Master of Technology

in

Computer Science and Engineering

(Specialization: Software Engineering)

by

Rohit Kumar

(Roll No.- 212CS3372)

under the supervision of

Prof. P. K. Sa

Department of Computer Science and Engineering
National Institute of Technology Rourkela
Rourkela, Odisha, 769 008, India

May 2014

Department of Computer Science and Engineering
National Institute of Technology Rourkela
Rourkela-769 008, Odisha, India.

Certificate

This is to certify that the work in the thesis entitled ***On the Use of Software Metrics*** by ***Rohit Kumar*** is a record of his research work carried out by him under my supervision and guidance in partial fulfillment of the requirements for the award of the degree of Master of Technology with specialization in Software Engineering in the department of Computer Science and Engineering, National Institute of Technology Rourkela.

NIT Rourkela
June 3, 2014

Pankaj Kumar Sa
Assistant Professor, CSE Department
NIT Rourkela, Odisha

Acknowledgment

I am grateful to numerous local and global peers who have contributed towards shaping this thesis. At the outset, I would like to express my sincere thanks to **Prof. Pankaj Kumar Sa** for his advice during my thesis work. As my supervisor, he has constantly encouraged me to remain focused on achieving my goal. His observations and comments helped me to establish the overall direction to the research and to move forward with investigation in depth. He has helped me greatly and been a source of knowledge.

I am very much indebted to **Prof. Santanu Ku. Rath**, Head-CSE, for his continuous encouragement and support. He is always ready to help with a smile. I am also thankful to all the professors at the department for their support.

I would like to thank all my friends and lab-mates for their encouragement and understanding. Their help can never be penned with words.

I must acknowledge the academic resources that I have got from NIT Rourkela. I would like to thank administrative and technical staff members of the Department who have been kind enough to advise and help in their respective roles.

Last, but not the least, I would like to dedicate this thesis to my family, for their love, patience, and understanding.

Rohit Kumar

Roll-212cs3372

Author's Declaration

I, **Rohit Kumar** (Roll No. **212CS3372**) understand that plagiarism is defined as any one or the combination of the following

1. Un-credited verbatim copying of individual sentences, paragraphs or illustrations (such as graphs, diagrams, etc.) from any source, published or unpublished, including the Internet sources.

2. Un-credited improper paraphrasing of pages or paragraphs (changing a few words or phrases, or rearranging the original sentence order).

3. Credited verbatim copying of a major portion of a paper (or thesis chapter) without clear delineation of who did or wrote what.

I have made sure that all the ideas, expressions, graphs, diagrams, etc., that are not a result of my work, are properly credited. Long phrases or sentences that had to be used verbatim from published literature have been clearly identified using quotation marks.

I affirm that no portion of my work can be considered as plagiarism and I take full responsibility if such a complaint occurs. I understand fully well that the guide of the thesis may not be in a position to check for the possibility of such incidences of plagiarism in this body of work.

Place: NIT Rourkela
Date: June 3, 2014

Rohit Kumar
Roll: 212CS3372
CSE Department(S/W Engg)
NIT Rourkela, Odisha

Abstract

The objective of any System software having a specific version in the present day scenario is to survive in the market for a considerably longer duration. In order to enhance the usability of existing software, the user has to understand its full functionality. For this a friendly graphical visualization is mandatory for its reliability and efficiency factor. In Software industry, there are so many SV Tool but end users do not know how to use all functionality effectively. As a result, a visualization of different modules at class level should be illustrated in order to provide better comprehension for end users. For this any software should be pictorially visualized dynamically according to end user requirement such that user can easily understand and reduce complexity according to their requirement.

In this thesis, it has been studied for visualizing the system complexity view of any Object -Oriented System to analyze which class is more complex and which class needs to be fragmented to reduce the redundancy. Software Metrics value has been calculated using several Java plugin and Software metrics are used as input to design a classifier using different Statistical technique to predict class as faulty and non faulty module.

Keywords: Software Visualization, System complexity, Software metrics, MooseFinder, K-Mean, Logistic.

Contents

List of Figures

List of Tables

List Of Abbreviation

SV	Software Visualization
CK	Chidamber & Kemerer
LOC	McCabes line count of code
V(g)	Cyclomatic Complexity
EV(g)	Essential Complexity
IV(g)	Design Complexity
FP	False Positive
FN	False Negative
TN	True Negative
TP	True Positive
NOM	Number of methods
NOA	Number of attributes
NOC	Number of classes
NOP	Number of Package
UML	Unified Modeling Language
NOMI	Number of methods inherited
NOPA	Number of private attribute
N$\bar{P}$OA	Number of public attribute
NOPM	Number of private methods
N$\bar{P}$OM	Number of public methods

CHAPTER 1

Introduction

Introduction

Literature Review

Various Performance Measures

Motivation

Problem statement

Thesis Organization

Chapter 1

Introduction

Program Visualization is the static or dynamic 2D or 3D visual representation of information about software system based on their structure, size, history or behaviour. Program Visualization is an approach to provide clear and deep understanding of existing software. Program Visualization uses visual representation to make software visible and reduces complexity.

Software metric data is used as an information for visualizing the different aspect of Software system.

Visualization concerns about graphical representation of software including structure and behavior of algorithms, program code and processed data. Program Visualization emphasize upon the technique to give physical shape to intangible or shapeless software.

Application of SV as-

- Algorithm Animation
- Software Engineering
- Concurrent Program Execution
- Static and Dynamic visualization of OO code
- Fault Diagnostics
- Debugging
- Requirement Analysis.

The concept of SV has transformed from 2 Dimensional to 3 Dimensional and then to Virtual Environments. 2 Dimensional SV represents a large number of nodes and arcs in tree like structure. In order to visualize such representation it presents pieces of graph in different windows. Different users can see different windows of same graph according to their requirements. But 2Dimensional visualization jumbles a large amount of information on a single plane. Due to this concept of 3 Dimensional came into picture.

1.1 Literature Review

This section presents a review of literature on the Software Visualization and its application in various field.

H.Childs et al., [1] analyzes several upcoming challenges and its corresponding future development in the field of Software Visualization as Massive Parallelisation,Processor Architectures and Programming Models, Application architecture and data management, Data Models, Rendering and Interaction. Denis et al., [2] analyzes different SV techniques for graphical visualization(static or dynamic)represenatation. It uses SV techniques as 2Dimensional Graphical representation, 3Dimensional Graphical representation and Virtual Environments with the help of Visual Metaphors.

Michele Lanza et al., [3, 4]emphasize upon evolution matrix to visualize the evolution of classes from one version of software to another version.It uses CodeCrawler tool and Moose Finder application to viualize system level view. T Mens et al., [5]focused on classification of predictive analysis(before evolution) and retrospective analysis(after evolution) where software metrics value can be used to improve software quality. Predictive analysis deals with Evolution-critical parts,Evolution-prone parts and Evolution Sensitive parts.Retrospective analysis deals with analysing the software and anlyasing the process.

JI Maletic et al., [6] emphasize upon taxonomy of Software Visualization systems.It also maps different SV system to different dimension of framework given in 1.1 and said that a single SV tool [2] can't address all visualization tasks.It

uses different visualization technique based on the SV dimension. N Hanakawa [7]proposed 3D visualization using module and logical coupling.For visualizing the software complexity,visualization tool maps metric value to module couping map on time-series.

S Bassil et al. [8]analyze a large number of Visualized Software tools(using Object-oriented language,procedural language, declarative language etc) .It addresses a functional aspect,cognitive aspect and code analysis of SV tools. C.Upson et al., [9] focused at developing an interactive application having high computational requirement and interactive graphics for programmers.

G.Abram et al., [10] analyzes about data visulaization through Data flow architecture model with the help of programming paradigm. H.Childs et al., [11]discussed about VisIt,an end user tool for visualizing large data. C Couto et al., [12] discover bugs on the basis of software bugs prediction and the occurence of bugs.It uses Granger Test to predict bugs with the help of 6 CK Metrics and other 11 metrics.

C Couto et al., [13] gave a dataset of the 17 Object Oriented metrics in a time-series fashion with an aim on source code evolution.Seventeen Object Oriented metrics include NOA, NOM, LOC, Coupling Metrics and CK Metrics. J E Retna et al. [14] analyzes upon several Software Quality Paramter and the parameter used for measuring the qualities of Software. S H Brown [15] implemented the Linear Regresseion and Multiple Linear Regression analysis with the method of finding Least Squares to find linear relationship between independent and dependent variables.

1.2 Various Performance Measures

1.2.1 Dataset Used

Dataset 1.2 used for fault diagnosis using various statistical method like Linear regression analysis, Polynomial regression analysis, logistic regression analysis, Naive Bayes and K-means clustering is taken from Promise Software Engineering Repository [16] to predict the fault.Dataset 1.2 is available publicly to perform

research in the field of Software Engineering domain.

Table 1.1: Mapping of five SV tool against five dimension [6]

SV System	**Task**	**Audience**	**Target**	**Representation**	textbfMedium
SHriMP	Reverse engineering, maintenance	Expert developer	Source code, documentation, static design-level information, medium Java systems	2D graphs, interactive, drill-down	color monitor
Tarantula	Testing, defect location	Expert developer	Source code, test suite data, error location	Line oriented representation, color interactive, filtering, selection	Color monitor
IMSOvison	Development, reverse engineering, management	Expert developer, team manager	Source code, static design information, metrics, large OO systems	Specialized visual language, 3D color objects, spatial relationships	Immersive virtual environment
Seesoft	Fault location, maintenance, reengineering	Expert developer	Source code, execution data,historical data	Line oriented represenataion, color, interactive, filtering, selection	Color monitor
Jinsight	Optimization	Expert developer	Program bursts, Java, dynamically collected	Color coded line oriented, text, interactive	color monitor

1.3 Motivation

In Software system, there are various parameters which are not uniquely referenced to one another.For example-Software product lines usually contain a hundred of variation points. Each variation point might have several other redundant information such as defining or binding another variation point. Generally, visualizing the static 2 Dimensional image is easy, but when we need to visualize 3 Dimensional, animated data or interactive manipulative data, then what we need to do. When we have to deal with large data set on multiple cores on a single computer or across multiple compute nodes, then we need to visualize that whole set of data in a pictorial representation for better human comprehension.

1.4 Problem Definition

Software Visualization is the graphical or a visual understanding of software systems.Nowadays,the software system are extremely complex and very difficult to operate as there is a huge amount of data. In order to cope with huge amount of data and its perplex distribution within the software module are the great challenging task for software developer.

1.5 Thesis Organization

The rest of the thesis is organized as follows.
Software Evolution using a combination of Software Visualization and Software metrics techniques is given in **Chapter-2**, in which a project consisting of more than 60 classes are visualized through a MOOSE application with the help of VerveineJ external exporter.Software system complexity view can easily be visualized through MooseFinder model.Then after metrics value has calculated through Eclipse PLUGIN and implemented, then their values are compared to predict the fault of software metrics using different statistical method.

Chapter 3 presents the several statistical methods like Linear Regression, Polynomial regression, Logistic regression, Naive Bayes and K-Mean Clustering to compute the parameter Accuracy, Precision, Recall, Specificity with the help of Confusion Matrix. With the help of accuracy and other parameters, we can distinguish which class involving which parameter is more complex or less complex.

Table 1.2: Dataset Used for Fault Diagnosis

LOC	**V(g)**	**EV(g)**	**IV(g)**	**Defects**
22	3	1	3	0
19	37	21	26	1
12	1	1	1	1
3	1	1	1	0
7	2	1	1	0
3	1	1	1	0
33	6	1	5	1
92	4	1	4	1
44	7	1	7	1
14	13	1	12	1
7	1	1	1	0
9	3	1	1	0
3	1	1	1	0
30	4	1	2	0
5	1	1	1	0
41	73	30	41	1
13	3	1	3	1
16	4	1	2	0
5	1	1	1	0
40	10	1	8	0
5	1	1	1	0
79	12	4	6	1
57	9	1	7	1
49	7	1	5	1
58	10	5	1	1
5	1	1	1	0
12	2	1	1	0
31	5	1	5	0
12	15	3	12	1
13	20	9	10	1
28	3	1	3	1
22	6	1	3	0
20	4	1	4	0
92	4	1	4	1
14	13	1	12	1
19	37	21	26	1
27	6	3	5	0
6	1	1	1	0
71	10	8	9	1
15	2	1	2	1
33	6	1	5	1
14	3	1	2	0
31	4	1	2	1
29	5	1 7	3	1
12	1	1	1	1

Chapter 2

Software System Complexity View

Introduction

Methodology Used

Contributions

Implementation

Summary

Chapter 2

Software System Complexity View

2.1 Introduction

In present day scenario, dealing with huge amount of data in any software system is a major problem. When we need to visualize such huge data on a single plane or even in the multi-dimensional plane, it creates a lengthy view to the user and leads to a complex view of software. In order to reduce the complexity, we incorporate the new term software metrics with software visualization. Software metric is the measurement of some property of the system software or its specification. It helps in predicting the complexity of software. When we know about the exact value of different software metric, then only we can filter which module involving which class is relevant or not. For this, we need to view the different module within the same class through pictorial representation. That pictorial representation may be System level view, Class level view etc.

Now in our day to day life, different software are coming with their new innovative technology, then after few months another software comes with some new feature and later one software supersede with the existing software. Even changing from one version to another version of the same software requires too much information about which module is more consistent or which module contains redundant information. Based on the graphical information we can classify which class is more complex and which class needs refactoring into some more classes.

Understanding Software Evolution suggested by Michele Lanza [3,17] also dealt

with Software metric in combination with software visualization to show software evolution matrix. Evolution matrix deals with different version of the same software.It tried to show only which class has been removed or added from one version after another version through system level view presentation.

It also defines the size of the system, growth and stabilization phase of the system software.While representing the system using class metric NOA and NOM [18, 19], we can see which class grows from one version to another version. On the basis of class sizes(either grow, shrinks or remain the same) classes can be categorized into several categories:

- Dayfly Classes:Exist during only one version of the software.
- Persistent Classes:Remain in existence during all versions of the software.
- Pulsar :Grows or shrink over and over again during whole all versions of the software.
- Supernova:Suddenly explodes in size.
- White Dwarf:Shrink in size gradually from one version to another.
- Red Giant:Constant over all versions .
- IDLE classes:Remains unchangeable over all versions.

2.2 Methodology Used

2.2.1 Moose Application

According to the Stephane Ducasse [20], Moose is a language independent FAMIX meta-model [21, 22]. FAMIX is a model which helps in supporting the presentation of different Object-Oriented Language.Moose is also extensible in nature as we don't know the future requirement of any problem specific environment. Moose is an open-source generic platform for users to analyze the systems software and their corresponding data.The data which are inputted to the system is mainly the source code . Through source code, the application can evaluate all the properties

and the relations within data.Source code may be written in ObjectOriented language or some meta-data of the software. This source code is fetched into Moose application through several importers. We can import data from different sources and in different formats. For example- Moose application can import the whole system software written in various programming languages either through internal importers (e.g., Smalltalk, XML, MSE), or through an external importer (e.g., Java, JEE, C++).

2.2.2 MooseFinder

MooseFinder [20] is a specific type of tool which works in support with Moose platform.It was created by Tudor Girba. It creates different queries based on requirement of different entities and their relationship. Created Queries may involve another query to understand the different criteria.Basically, it helps in importing different independent language to form a model and that model is viewed pictorially with the help of external plug in like VerveineJ(external Java parser, that exports MSE model).

2.3 Contributions

Implementation and analysis of the Java source code with more than 60 classes as a project(project name-over˙all) to find its system complexity view through MOOSE Finder application with the help VerveineJ exporter. It visualizes with the help of a number of attributes NOA, number of methods NOM, number of classes NOC and number of Packages NOP as discussed in [20]. We can also visualize through several pictorial views like class level Blueprint, Cloud view, UML view and so on.

Steps in System Complexity View

1. Create a Java project having minimum 60 classes.

2. Find out .Class file of all classes in a particular Java specific folder named as over˙all.

3. Open a Pharo image file that will load MSE application as default file into its own platform.

4. Select the model from the initial list and validate it.For example- For Java project,we have to select FAMIX-Java model from the selection list 2.4 .

5. Perform next and import the name of the model which we want to visualize in various views given in figure2.1.

6. Choose ClassPackageImporter to import all its contents, such as attribute,method ,package,class and so on given in figure2.2.

7. After uploading the over˙all model go to All Classes-All famixattributes given in figure2.3 .

8. Choose Visualize System Complexity view and it will generate given in figure2.4.

Steps for finding Metrics Value

1. Go to Java Eclipse(Juno) to load workbench in suitable path.

2. Go to File and open Project in Java perspective.

3. After running the Java file successfully, it creates .class file.

4. Go to Window menu Select Preferences and choose Metric Preferences .

5. Go to Window menu, then "Show view" option, then "Others".Finally, select Metrics - Metrics View shown in figure 2.5.

6. After selecting the Metrics view ,the metric value will be shown given in figure2.6

2.4 Implementation

To implement the above work, a Java project of at least 60 Java classes has been used. In this section to find the System complexity view and then Software Metrics value using MooseFinder Application and Java Eclipse Plugin have been shown.

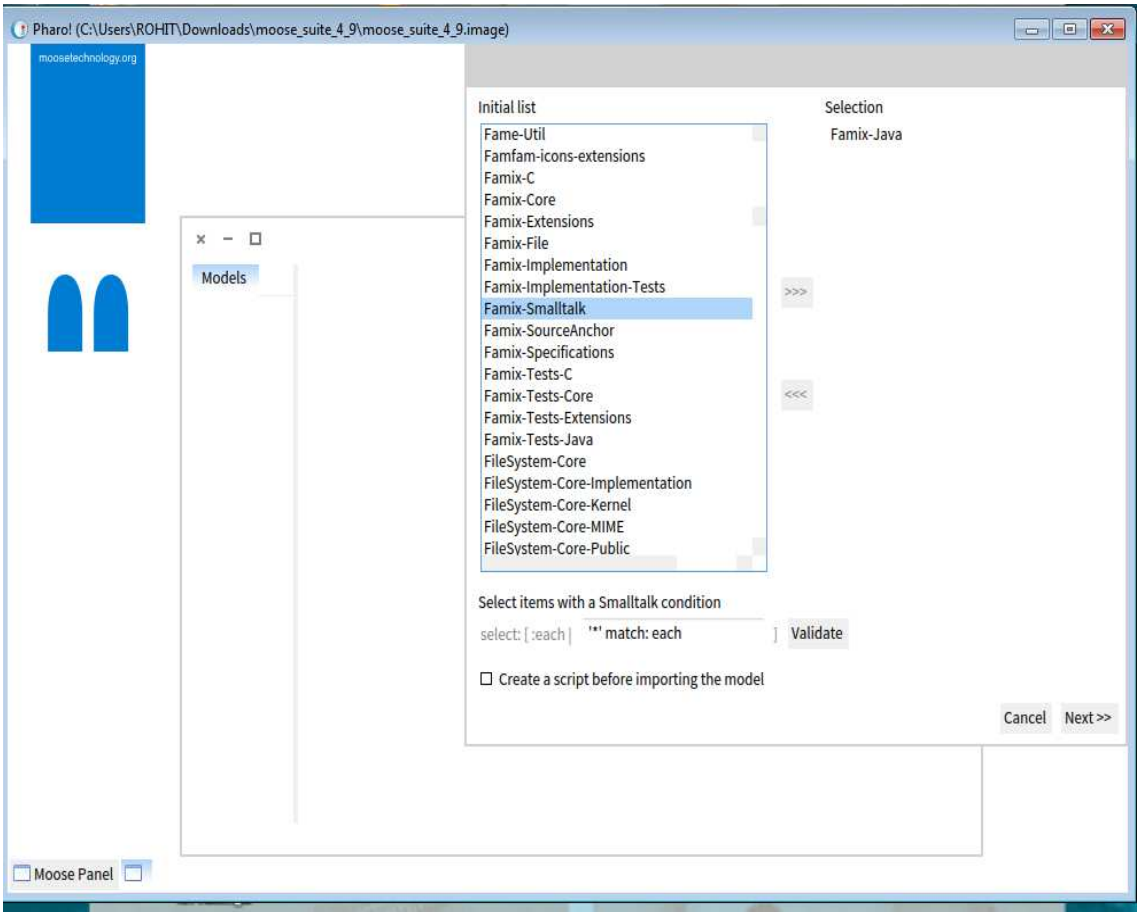


Figure 2.1: Select script Famix Java for executing Model

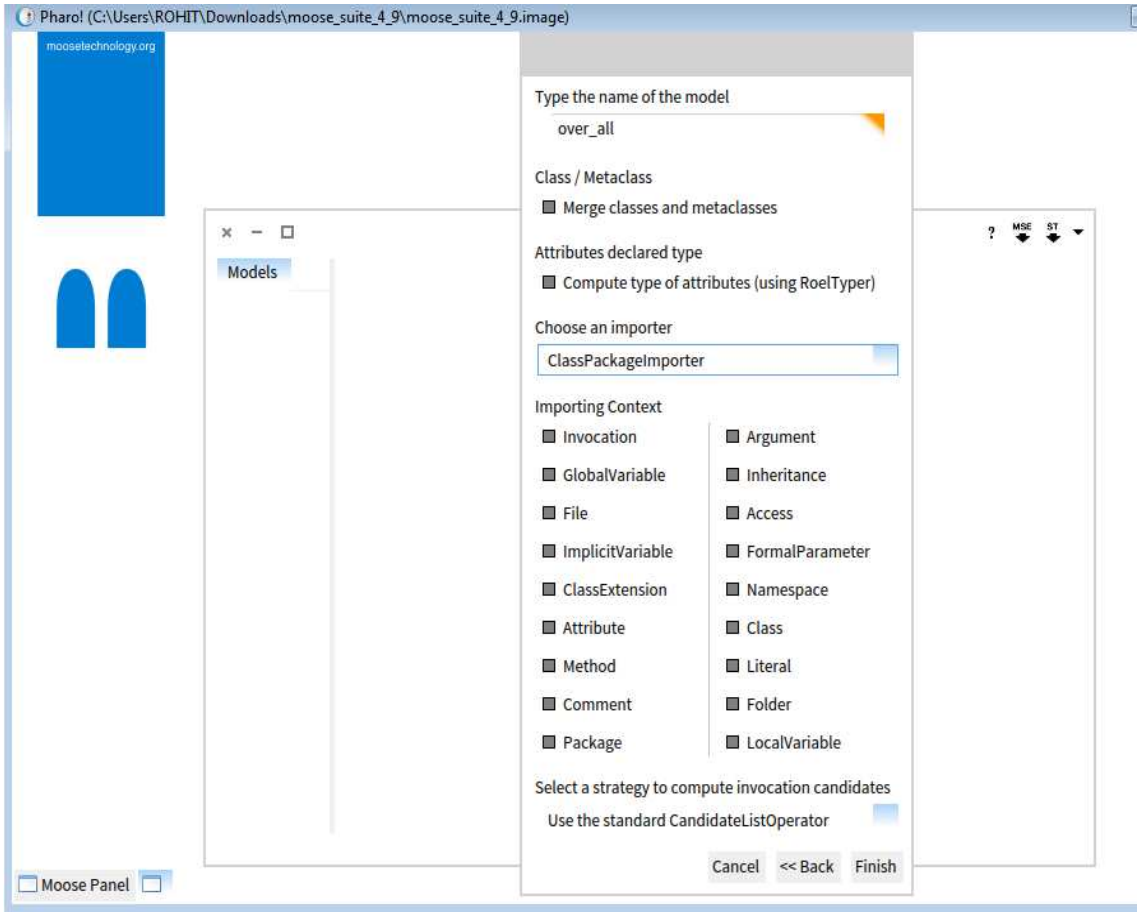


Figure 2.2: Import the required model under Moose Platform

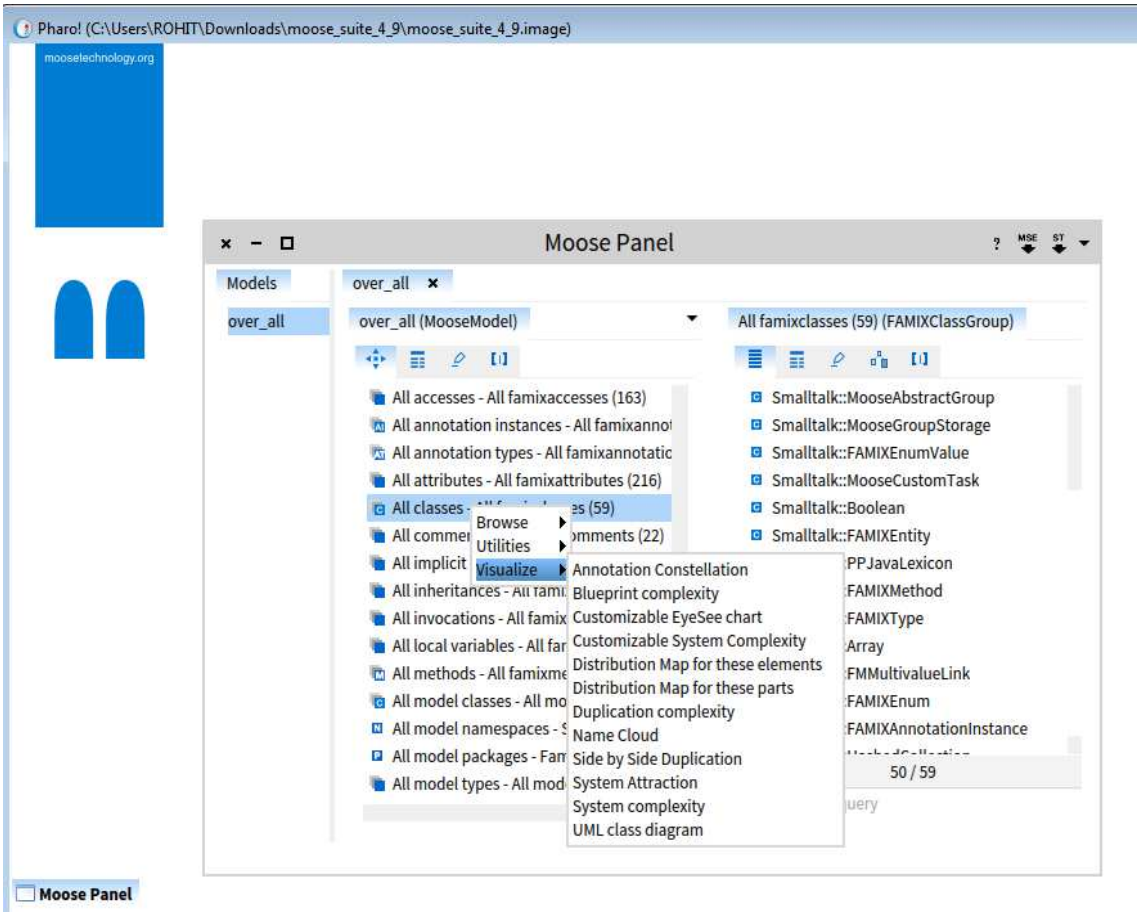


Figure 2.3: Selection of All Classes under Moose Panel

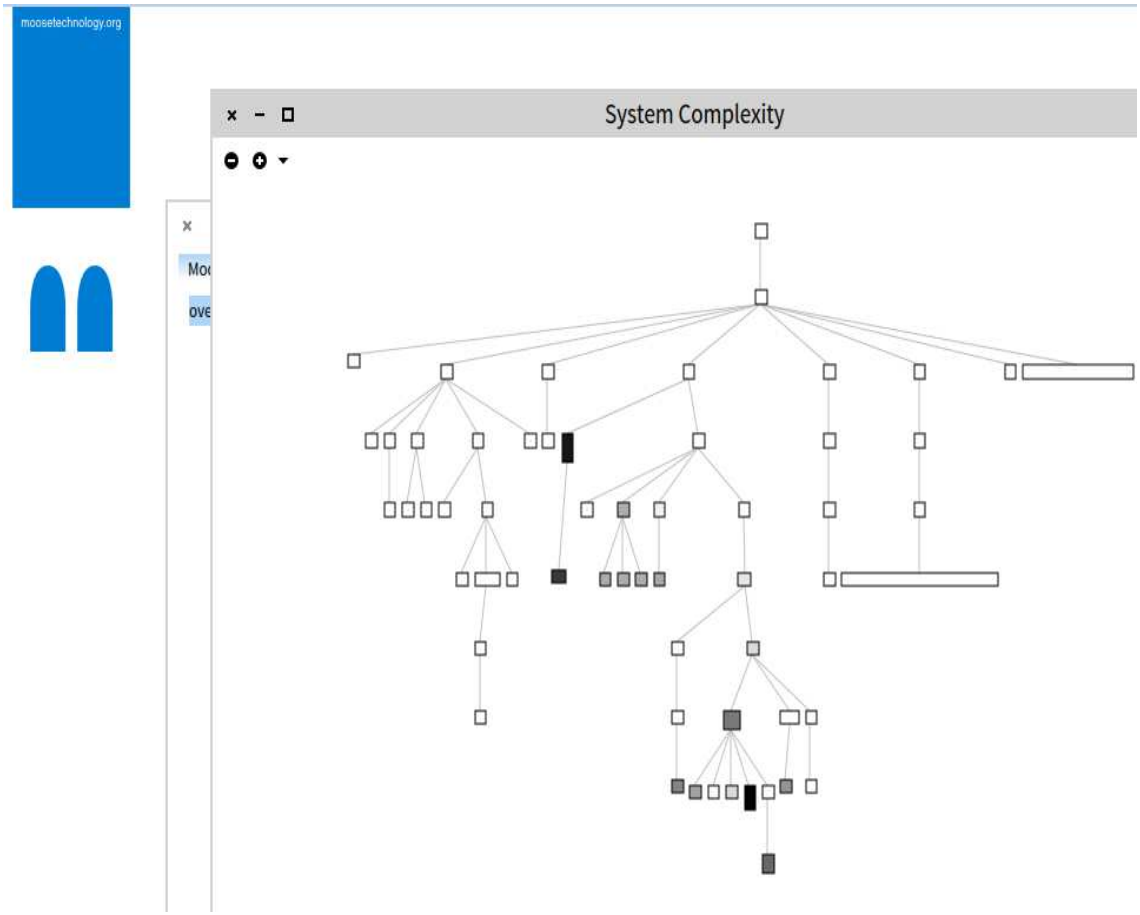


Figure 2.4: System Complexity View

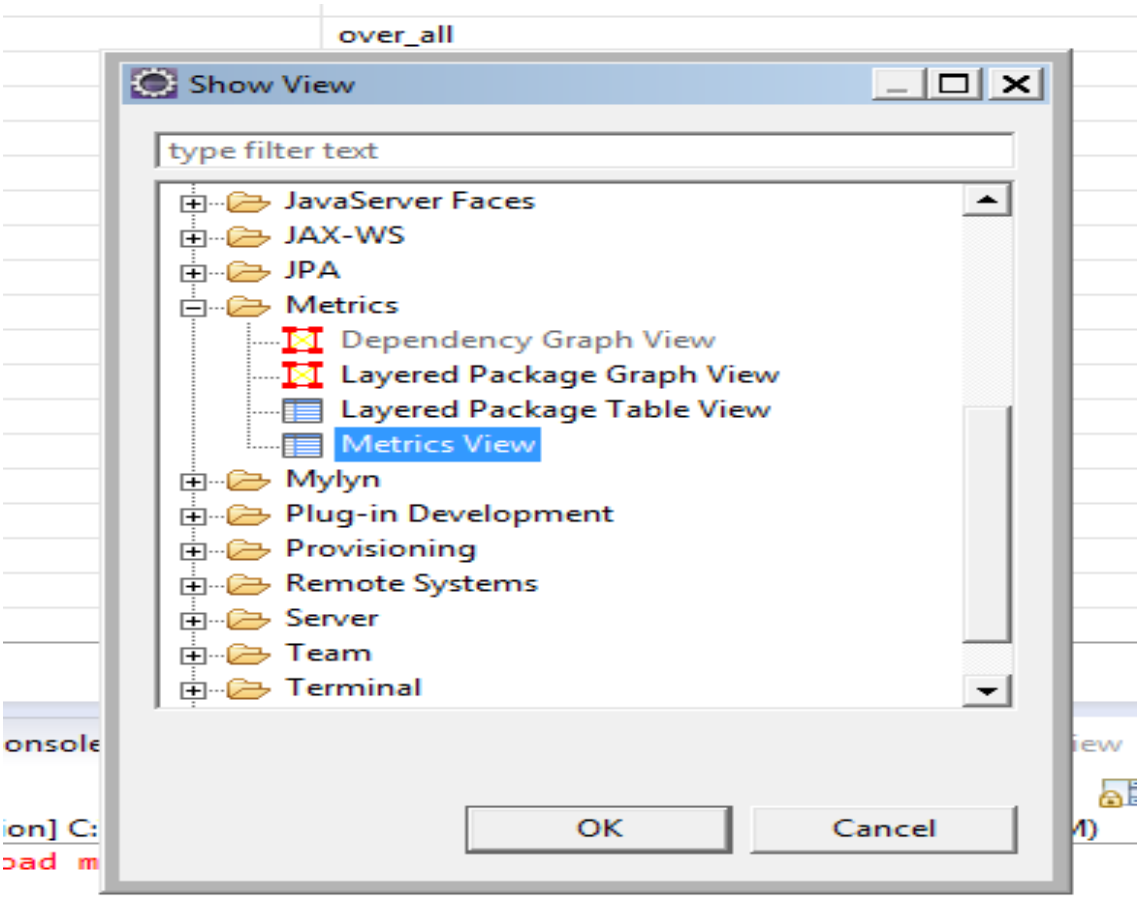
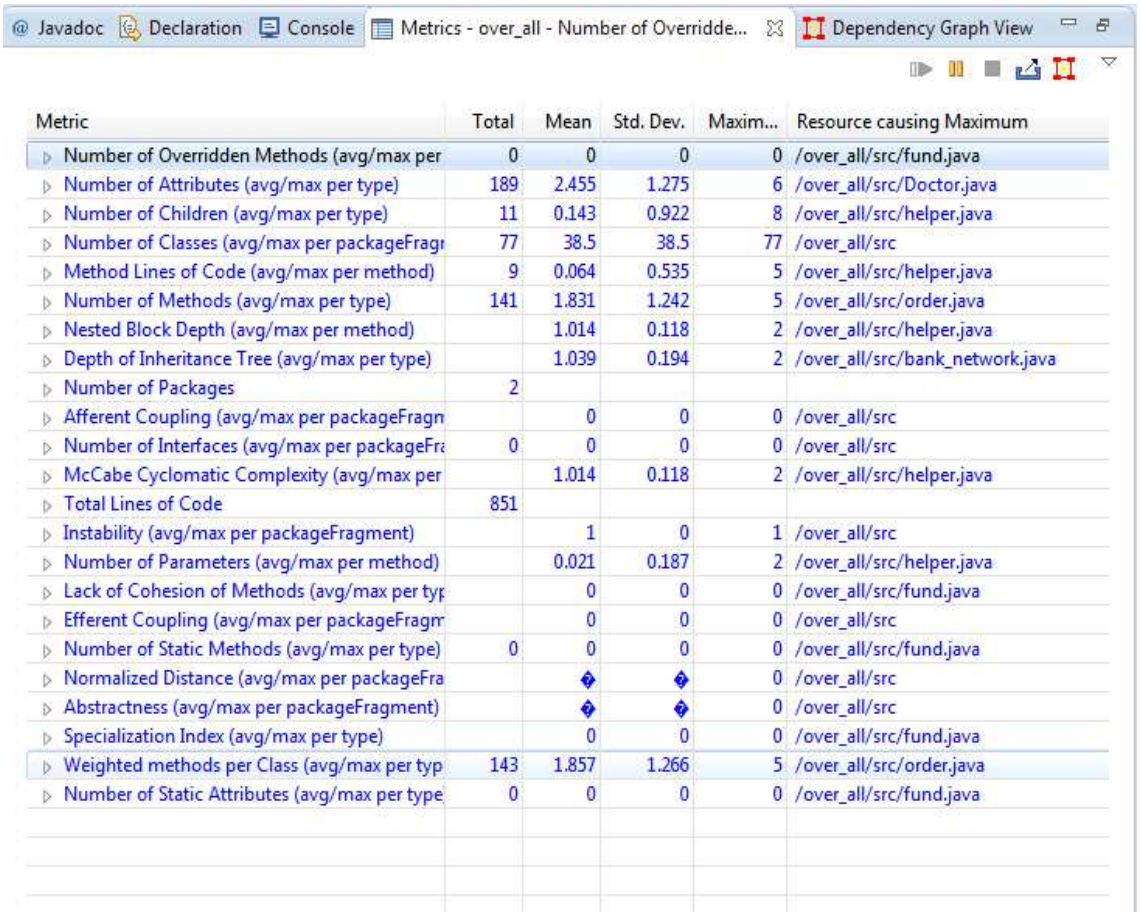


Figure 2.5: Select Metrics View under Show View



Metric	Total	Mean	Std. Dev.	Maxim...	Resource causing Maximum
Number of Overridden Methods (avg/max per	0	0	0	0	/over_all/src/fund.java
Number of Attributes (avg/max per type)	189	2.455	1.275	6	/over_all/src/Doctor.java
Number of Children (avg/max per type)	11	0.143	0.922	8	/over_all/src/helper.java
Number of Classes (avg/max per packageFragr	77	38.5	38.5	77	/over_all/src
Method Lines of Code (avg/max per method)	9	0.064	0.535	5	/over_all/src/helper.java
Number of Methods (avg/max per type)	141	1.831	1.242	5	/over_all/src/order.java
Nested Block Depth (avg/max per method)		1.014	0.118	2	/over_all/src/helper.java
Depth of Inheritance Tree (avg/max per type)		1.039	0.194	2	/over_all/src/bank_network.java
Number of Packages	2				
Afferent Coupling (avg/max per packageFragn		0	0	0	/over_all/src
Number of Interfaces (avg/max per packageFra	0	0	0	0	/over_all/src
McCabe Cyclomatic Complexity (avg/max per		1.014	0.118	2	/over_all/src/helper.java
Total Lines of Code	851				
Instability (avg/max per packageFragment)		1	0	1	/over_all/src
Number of Parameters (avg/max per method)		0.021	0.187	2	/over_all/src/helper.java
Lack of Cohesion of Methods (avg/max per typ		0	0	0	/over_all/src/fund.java
Efferent Coupling (avg/max per packageFragm		0	0	0	/over_all/src
Number of Static Methods (avg/max per type)	0	0	0	0	/over_all/src/fund.java
Normalized Distance (avg/max per packageFra		?	?	0	/over_all/src
Abstractness (avg/max per packageFragment)		?	?	0	/over_all/src
Specialization Index (avg/max per type)		0	0	0	/over_all/src/fund.java
Weighted methods per Class (avg/max per typ	143	1.857	1.266	5	/over_all/src/order.java
Number of Static Attributes (avg/max per type	0	0	0	0	/over_all/src/fund.java

Figure 2.6: Metrics Value in terms of various parameter

2.5 Conclusions

System Complexity View using different Software visualization techniques is a fast growing research area in the field of visualizing different aspects of software either in terms of class level or through module level.Software visualization is now a days became the most important aspect in order to assist our human brain to understand the complexity level of the different paradigm of advance software.

As Software aspect changes from one version to another, we need to pictorially visualize in several dimensions(2D, 3D or virtual environment) in order to understand which part of the software is more complex or which part of the software is constantly redundant code and needs to be removed. Even we can analyze through Class level view from one version to another version and check whether one class shrinks ,expand or remains the same in terms of NOA and NOM.

In this chapter,a MooseFinder application with the help of Pharo 1.4J Smalltalk environment is used to visualize the System complexity view of a Java project of more than 60 classes.

After getting the System Complexity View, finding metrics value in terms of NOM, NOA, NOC, NOP, Total Lines of Code with the help of Java Eclipse plugin. Now next chapter deals with how to find a particular class is more complex or which class is less complex if we have provided with software metrics value in terms of NOA, NOM.

For this we apply several statistical techniques to classify among these classes with metric value analysis to find accuracy, precision, etc.

Chapter 3

Fault Diagnosis Using Statistical Method

Chapter 3

Fault Diagnosis Using Statistical Method

3.1 Introduction

The objective of any software product developed in the present day scenario needs to establish and survive in the market for a longer duration. In order to achieve these objectives, new functionalities need to be added to the existing software. This increase in functionality and it leads to program becomes more complex and larger in size. As the size of the program become more, it means either number of classes will go on increase or the number of modules within the classes will increase. In this thesis, an attempt has been made to diagnose the fault with the help of five statistical technique. It is observed that numerous software metrics are being used as input for estimation. In this study, Software metrics has been considered to provide requisite input data to estimate the fault and find the confusion matrix parameter. This approach is applied using several statistical technique. The performance of this approach is evaluated based on the various parameters available in literature such as: Accuracy, Precision, Correctness, Completeness.

3.2 Methodology Used

The following sub-sections highlight on the various statistical methods used for finding which class is more complex or not.

3.2.1 Linear Regression Analysis

Linear regression is one of the fundamental statistical technique .It is used to determine the relationship between an independent variable and a dependent variable.It is the study of linear (i.e., straight-line) relationship between variables.

The Univariate linear regression is based on:

$$Y = \beta_0 + \beta_1 X_i, i = 1, 2, 3, ..., n \tag{3.1}$$

where Y is dependent variable, X is independent variable and β_0, β_1 are the constant and coefficient values.

In case of multivariate or multiple linear regression, the linear regression is based on:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + ... + \beta_p X_p \tag{3.2}$$

Where X_i are the independent variable and y is dependent variable.

The method of Least Square [23] is widely used to find a linear relationship(straight line) between variables that best fits the data.

In order to find the regression coefficient ,Least square method [24] is used in 3.1 3.2 by estimating

$$\beta = (X^T X)^{-1} X^T Y. \tag{3.3}$$

3.2.2 Polynomial regression analysis

Polynomial regression is one of the widely used statistical technique in extension to linear regression. In polynomial regression analysis, the relationship between the independent variable x and the dependent variable y is represented as an nth order polynomial defined in equation3.4.

Polynomial regression best fits a nonlinear or curvilinear relationship between the independent value of x and the dependent value of y. Polynomial models are useful in those situations where output are curvilinear in original function [25]. Polynomial models are also useful as very complex nonlinear relationship can be approximated by this technique to small range of xˆi.

The Univariate Polynomial regression analysis is based on:

$$Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3 + ... + \beta_n X^n \tag{3.4}$$

where Y is dependent variable, X is independent variable and β_0, $\beta_1 ... \beta_n$ are the constant and coefficient values.

Equation 3.4 show the Univariate Polynomial regression of n order polynomial. In this report, second order polynomial is considered for finding the relationship between fault in terms of various parameter and metrics of the class. The second order polynomial is represented as:

$$Y = \beta_0 + \beta_1 X + \beta_2 X^2 \tag{3.5}$$

In case of multivariate second order Polynomial regression analysis, the Polynomial regression of two variable is based on:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_{11} X_1^2 + \beta_{22} X_2^2 + \beta_{12} X_1 X_2 \tag{3.6}$$

3.2.3 Logistic regression analysis

Logistic regression is the commonly used probabilistic statistical technique,also known as binary logistic regression. It is a kind of regression analysis used for predicting the outcomes of dependent variable based on one or more independent variables. A dependent variable can have only two possible values(binary outcome) either zero or one.

The form of this relationship is as follows:

$$P(x) = \frac{e^{\alpha+\beta X}}{1 + e^{\alpha+\beta X}} \tag{3.7}$$

$$P(x) = \frac{e^{L}}{1 + e^{L}} \tag{3.8}$$

where P is logistic function defines the probability of an event and increases from 0 to 1 , X is the independent variable And L is a linear function of X defined as $L = \alpha + \beta X$.

Like other regression technique, logistic regression make use of one or more independent variables that can be either numerical or categorical data.

So the dependent variable of a class containing fault is divided into two groups, one group containing zero fault and the other having at least one fault. Logistic regression is used to construct a prediction model for the fault proneness of classes. In this method, metrics are used in combination. The multiple logistic regression model is based on the following equation:

$$logit[\pi(x)] = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + ... + \beta_p X_p \tag{3.9}$$

where x_i is the independent variable and $logit[\pi(x)]$ is a dependent variable. It shows that logistic regression analysis is a standard linear regression model and the Dichotomous(indicator variable) outcome in result is transformed by the *logit* transform. This transform changes the range of $\pi(x)$ from 0 to 1 to $-\infty$ to $+\infty$, as usual for linear regression. P represents the number of independent variables. π represents the probability of fault in the class during validation. It is defined as:

$$\pi(x) = \frac{e^{\beta_0+\beta_1 X_1+\beta_2 X_2+\beta_3 X_3+.....+\beta_p X_p}}{1 + e^{\beta_0+\beta_1 X_1+\beta_2 X_2+\beta_3 X_3+.....+\beta_p X_p}} \tag{3.10}$$

3.2.4 Naive Bayes

A Bayes classifier is a simple probabilistic classifier based on applying Bayes' theorem (from Bayesian statistics) with strong independence assumptions. A more descriptive term for the underlying probability model would be "independent feature model".

Naive Bayes classifier also known as Bayesian classification is based on Bayes' theorem. It assumes that all the features are independent and will not influence the estimation process. Naive Bayes classifier assigns the given object x to class $c^* = argmax_c P(c|x)$ by using $Bayes'$ rule given below:

$$P(c|x) = \frac{P(x|c)P(c)}{P(x)} \tag{3.11}$$

where $P(c)$, is the prior probability of a parameter c before having seen the

data. $P(c|x)$ is called the likelihood. It is the probability of the data x and define as

$$P(x|c) = \prod_{k=1}^{n} P(x_k|c) \tag{3.12}$$

3.2.5 K-Mean Clustering

K-Means is a hard C-means clustering algorithm based on finding data clusters in a data set. K-Means clustering algorithm clusters 'n' objects into 'k' partitions based on attributes such that k < n. The distance from each object to the centroid is computed using the Euclidean distance concept. The K-means algorithm is inherently iterative, and no guarantee can be made that it will converge to an optimum solution. The performance of the K-means algorithm depends on the initial positions of the cluster centers, thus it is advisable to run the algorithm several times, each with a different set of initial cluster centers. The steps followed in K-means clustering is as shown in Figure 3.1. Equation 3.13 shows the function for computing the Euclidean distance.

$$d(x, y) = \sum_{i=1}^{p} |x_i - y_i| \tag{3.13}$$

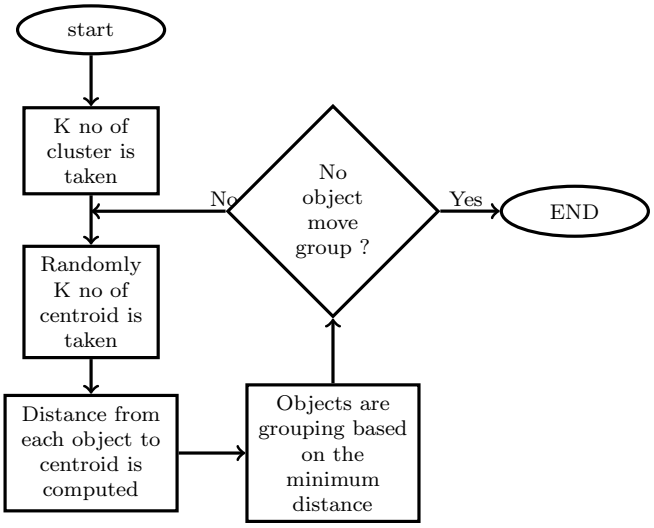


Figure 3.1: Flow chart K-Means clustering algorithm

3.3 Performance Evaluation Parameters

The performance parameters for statistical analysis can be determined based on the confusion matrix(also known as contingency matrix) as shown in Table 3.1. Confusion matrix is a table layout that allows visualization of performance of any supervised learning algorithm.

Table 3.1: Confusion matrix to classify a class as faulty and not-faulty

	NO(Prediction)	YES(Prediction)
NO (Actual)	True Negative (TN)	False Positive (FP)
YES(Actual)	False Negative (FN)	True Positive (TP)

The following are the performance measures used in classification.

- Precision

  It is defined as the degree to which the repeated measurements under unchanged conditions shows the same results.

$$Precision = \frac{TP}{TP + FP} \tag{3.14}$$

- Recall

  It is also known as Sensitivity indicates the how many of the relevant items to be identified.

$$Recall = \frac{TP}{TP + FN} \tag{3.15}$$

- Specificity

  Specificity focus on how effectively a classifier identifies the negative labels.

$$Specificity = \frac{TN}{FP + TN} \tag{3.16}$$

- Accuracy

  Accuracy measure is the proportion of predicted fault prone modules that are inspected out of all modules. It is defined as:

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{3.17}$$

3.4 Result

In this section, the relationship between value of metrics and the number of fault found among the class is determined.

Table 3.2 to Table 3.6 show the classification matrix for data set for the applied techniques such as linear regression, polynomial regression, logistic regression, navies bayes and K-Means clustering.

- From Table 3.2, it can be observed that totally 235 (209+26) classes were classified correctly with an accuracy rate of 76.80% when linear regression was applied.
- From Table 3.3, it can be observed that totally 242 (218+24) classes were classified correctly with an accuracy rate of 79.08% when polynomial regression was applied.
- From Table 3.4, it can be observed that totally 257 (251+6) classes were classified correctly with an accuracy rate of 84.29% when logistic regression was applied.
- From Table 3.5, it can be observed that totally 246 (233+13) classes were classified correctly with an accuracy rate of 80.39% when navies bayes was applied.
- From Table 3.6, it can be observed that totally 248 (218+30) classes were classified correctly with an accuracy rate of 81.05% when K-Means clustering was applied.

Table 3.2: After applying Linear Regression

	Not-Faulty	Faulty
Not-Faulty	209	49
Faulty	22	26

Table 3.3: After applying Polynomial Regression

	Not-Faulty	Faulty
Not-Faulty	218	40
Faulty	24	24

Table 3.4: After applying Logistic Regression

	Not-Faulty	Faulty
Not-Faulty	251	7
Faulty	42	6

Table 3.5: After applying Naive Bayes

	Not-Faulty	Faulty
Not-Faulty	233	25
Faulty	35	13

Table 3.6: After applying K-Means clustering

	Not-Faulty	Faulty
Not-Faulty	218	40
Faulty	18	30

Table 3.7, lists the values of obtained performance parameters for data set for the applied techniques. From Table 3.7, it is clear that logistics regression achieve better accuracy as compare to other techniques.

Table 3.7: Performance of Dataset 1.2

Technique	Precision	Recall	Specificity	Accuracy
Linear regression	0.8101	0.3467	0.5417	76.80
Polynomial regression	0.8450	0.3750	0.5000	79.08
Logistics regression	0.9735	0.4615	0.1250	84.29
Naives Bayes	0.9031	0.3421	0.2708	80.39
K-Means Clustering	0.8450	0.4286	0.6250	81.05

3.5 Summary

In this chapter, an attempt has been made to use software metrics in order to classify the dataset on the basis of several performance evaluation parameters.

Five different statistical analysis has been applied to promise dataset having different metrics value in terms of LOC ,V(g),EV(g),IV(g).Accuracy and precision value are considered for estimating the complexity level of classes.These statistical techniques have the ability to predict the output based on promised data. The software metrics from promise repository data set are taken as input data to train the data and estimate the fault involved. From this analysis, it can be concluded that Logistic Regression statistical technique gives the better result when compared to other statistical technique. Also, this comparative analysis highlights that there is a strong relation between software metrics and their accuracy value.

Further, work can be extended in removing the complex classes or module after identifying it through statistical technique.

Chapter 4

Conclusion and Future Work

Conclusion

Future Work

Chapter 4

Conclusion and Future Work

4.1 Conclusion

Several approaches have already been defined in the literature in the field of Software Visualization. However, to visualize the complexity view of Object-oriented system as a whole system is a great scope to understand its functionality through several SV tools like Code-crawler, Moose Finder, etc. Moose Finder is one of the open source SV tool to visualize various aspect of visualization like UML view , Cloud view, Blueprint view and System complexity view.

Then different statistical method has been applied over data set having four software metric parameter to find confusion matrix or contingency matrix. Through error matrix, predict the class based on actual class to find out the accuracy, precision, recall, specificity. The results are summarized in Table-3.7. The computations for above procedure have been implemented using MATLAB. Out of five statistical method, Logistic regression analysis give higher accuracy in comparison to other method.

This approach can also be extended by using other AI techniques such as genetic algorithm (GA), Artificial Neural Network (ANN) to achieve higher level of accuracy and precision

Finally, the generated results of different statistical method have been compared for estimating the performance of different software metric value. Hence it can be concluded that the fault detection using the Logistic Regression analysis will provide more accurate results than other statistical techniques.

4.2 Future Work

As complexity in software, computer graphics and massive parallelization are increasing significantly, textual means are too devastating for the programmer. Software Visualization approach allows us to easily understand the system and even reverse engineer it.

Visualizing the dynamic aspects of the module in a class is also needed during run time environment. Visualizing the parallelization across several cores within a computer or multiple node in distributed environment results in high speed computing and efficient performance.

In the field of software security, concept of Software visualization can be applied in order to trace the dependency between various data and analyze it through pictorial representation. The technique of SV can also be applied in Data mining through visualizing different data patterns.

Bibliography

[1] H. Childs, B. Geveci, W. Schroeder, J. Meredith, K. Moreland, C. Sewell, T. Kuhlen, and E. Bethel, "Research challenges for visualization software," *Computer*, vol. 46, no. 5, pp. 34–42, 2013.

[2] D. Gračanin, K. Matković, and M. Eltoweissy, "Software visualization," *Innovations in Systems and Software Engineering*, vol. 1, no. 2, pp. 221–230, 2005.

[3] M. Lanza and S. Ducasse, "Understanding software evolution using a combination of software visualization and software metrics," in *In Proceedings of LMO 2002 (Langages et Modles Objets*, pp. 135–149, Lavoisier, 2002.

[4] M. Lanza, "The evolution matrix: Recovering software evolution using software visualization techniques," in *Proceedings of the 4th international workshop on principles of software evolution*, pp. 37–42, ACM, 2001.

[5] T. Mens and S. Demeyer, "Future trends in software evolution metrics," in *Proceedings of the 4th international workshop on Principles of software evolution*, pp. 83–86, ACM, 2001.

[6] J. I. Maletic, A. Marcus, and M. L. Collard, "A task oriented view of software visualization," in *Visualizing Software for Understanding and Analysis, 2002. Proceedings. First International Workshop on*, pp. 32–40, IEEE, 2002.

[7] N. Hanakawa, "Visualization for software evolution based on logical coupling and module coupling," in *Software Engineering Conference, 2007. APSEC 2007. 14th Asia-Pacific*, pp. 214–221, IEEE, 2007.

[8] S. Bassil and R. K. Keller, "Software visualization tools: Survey and analysis," in *Program Comprehension, 2001. IWPC 2001. Proceedings. 9th International Workshop on*, pp. 7–17, IEEE, 2001.

[9] C. Upson, T. A. Faulhaber Jr, D. Kamins, D. Laidlaw, D. Schlegel, J. Vroom, R. Gurwitz, and A. Van Dam, "The application visualization system: A computational environment for scientific visualization," *Computer Graphics and Applications, IEEE*, vol. 9, no. 4, pp. 30–42, 1989.

[10] G. Abram and L. Treinish, "An extended data-flow architecture for data analysis and visualization," in *Proceedings of the 6th conference on Visualization'95*, p. 263, IEEE Computer Society, 1995.

[11] H. Childs, "Visit: An end-user tool for visualizing and analyzing very large data," 2013.

[12] C. Couto, C. Silva, M. T. Valente, R. Bigonha, and N. Anquetil, "Uncovering causal relationships between software metrics and bugs," in *Software Maintenance and Reengineering (CSMR), 2012 16th European Conference on*, pp. 223–232, IEEE, 2012.

[13] C. Couto, C. Maffort, R. Garcia, and M. T. Valente, "Comets: A dataset for empirical research on software evolution using source code metrics and time series analysis," *ACM SIGSOFT Software Engineering Notes*, vol. 38, no. 1, pp. 1–3, 2013.

[14] J. E. Retna, G. Varghese, M. Soosaiya, and S. Joseph, "A study on quality parameters of software and the metrics for evaluation," *International journal of Computer Engineering & Technology (IJCET), ISSN Print: ISSN*, pp. 0976–6367, 2010.

[15] S. H. Brown, "Multiple linear regression analysis: a matrix approach with matlab," *Alabama Journal of Mathematics*, 2009.

[16] J. Sayyad Shirabad and T. Menzies, "The PROMISE Repository of Software Engineering Databases.." School of Information Technology and Engineering, University of Ottawa, Canada, 2005.

[17] M. Lanza, "The evolution matrix: Recovering software evolution using software visualization techniques," in *Proceedings of the 4th International Workshop on Principles of Software Evolution*, IWPSE '01, (New York, NY, USA), pp. 37–42, ACM, 2001.

[18] M. Lanza, *Combining metrics and graphs for object oriented reverse engineering.* PhD thesis, Citeseer, 1999.

[19] S. Demeyer, S. Ducasse, and M. Lanza, "A hybrid reverse engineering approach combining metrics and program visualisation," in *Reverse Engineering, 1999. Proceedings. Sixth Working Conference on*, pp. 175–186, IEEE, 1999.

[20] S. Ducasse, M. Lanza, and S. Tichelaar, "Moose: an extensible language-independent environment for reengineering object-oriented systems," in *Proceedings of the Second International Symposium on Constructing Software Engineering Tools (CoSET 2000)*, vol. 4, Citeseer, 2000.

[21] S. Tichelaar, S. Ducasse, and S. Demeyer, "Famix and xmi," in *Reverse Engineering, 2000. Proceedings. Seventh Working Conference on*, pp. 296–298, IEEE, 2000.

[22] S. Demeyer, S. Ducasse, and E. Tichelaar, "Why famix and not uml? uml shortcomings for coping with round-trip engineering," in *In Proceedings of¡¿, Fort Collins*, Citeseer, 1999.

[23] S. P. Gordon and F. S. Gordon, "Deriving the regression equation without using calculus.," *Mathematics and Computer Education*, vol. 38, no. 1, pp. 64–68, 2004.

[24] S. M. Scariano and M. Calzada, "Three perspectives on teaching least squares.," *Mathematics and Computer Education*, vol. 38, no. 3, pp. 255–264, 2004.

[25] D. C. Montgomery, E. A. Peck, and G. G. Vining, *Introduction to linear regression analysis*, vol. 821. John Wiley & Sons, 2012.