DEVELOPMENT OF TECHNIQUES AND MODELS FOR IMPROVING SOFTWARE QUALITY

By

ANKITA BANSAL
Roll No.: 2k11/Ph.D./Comp.Sc./04

Under the guidance of

Dr. Ruchika Malhotra
Associate Head and Assistant Professor,
Department of Software Engineering

Submitted in fulfillment of the requirements of the degree of Doctor of Philosophy to the

Delhi Technological University
(Formerly Delhi College of Engineering)
Shahbad Daulatpur, Main Bawana Road, Delhi 110042
November, 2015

DECLARATION

I, Ankita Bansal, Ph.D. student (Roll No. 2k11/Ph.D./Comp.Sc./04), hereby declare that the thesis entitled **“Development of Techniques and Models for Improving Software Quality”** which is being submitted for the award of the degree of Doctor of Philosophy in Computer Science Engineering, is a record of bonafide research work carried out by me in the Department of Computer Science Engineering, Delhi Technological University. I further declare that this work is based on original research and has not been submitted to any university or institution for any degree or diploma.

Date: ________________

Place: New Delhi

Ankita Bansal

2k11/Ph.D./Comp.Sc./04

Department Of Computer Science Engineering

Delhi Technological University (DTU)

New Delhi -110042

CERTIFICATE

DELHI TECHNOLOGICAL UNIVERSITY

(Govt. of National Capital Territory of Delhi) BAWANA ROAD, DELHI - 110042

Date: ___________________

This is to certify that the work embodied in the thesis titled **”DEVELOPMENT OF TECHNIQUES AND MODELS FOR IMPROVING SOFTWARE QUALITY”** has been completed by**Ankita Bansal** under the guidance of **Dr. Ruchika Malhotra** towards fulfillment of the requirements for the degree of Doctor of Philosophy of Delhi Technological University, Delhi. This work is based on original research and has not been submitted in full or in part for any other diploma or degree of any university.

Supervisor

DR. RUCHIKA MALHOTRA

Associate Head and Assistant Professor, Department of Software Engineering

Delhi Technological University, Delhi 110042

ACKNOWLEDGMENT

While submitting my thesis, titled as ”Development of Techniques and Models for Improving Software Quality” for the fulfillment of the requirements for the degree of Doctor of Philosophy, I feel that this is the time to acknowledge the contribution of my teachers, friends and relatives in completion of this mammoth task.

I consider myself fortunate and privileged to have Dr. Ruchika Malhotra as my supervisor. I am deeply indebted to her for shaping my path to research by guiding me with her deep knowledge and insightful discussions. I thank her for the valuable guidance, scholarly inputs and consistent inspiration which I received throughout the research work. I express my sincere gratitude and thanks to her, not only for the guidance for research but also for the personal and emotional support she gave me from time to time. Her moral support in times of need was crucial in carrying out this research in an effective manner. Dr. Ruchika Malhotra has deep knowledge of the subject and was available to me at every step when this research work was going on. Her cooperation, positive attitude and understanding really deserve an everlasting appreciation. I could not have wished for a better supervisor. I hope for and look forward to her continued guidance in future.

I am deeply indebted to Prof. Yogesh Singh, Vice Chancellor of Delhi Technological University and Director of Netaji Subhas Institute of Technology. He has been a constant source of inspiration for me throughout my research and teaching career. I am grateful to him for constantly encouraging and supporting me in this endeavour of mine.

I thank Prof. O.P. Verma, DRC Chairman, Delhi Technological University for his encouragement, help and cooperation throughout the work. I express my gratitude and thanks to Prof. S.K. Dhurandher, Head of the Department of Information Technology in Netaji Subhas Institute of Technology, for his kind support and help during the time of my thesis writing. I am sincerely thankful to Abha Jain, Research Scholar

(Delhi Technological University) for all her help and support in this research work. I acknowledge with thanks the contribution of all my other teachers and friends for their direct and indirect help and support in this work.

A special thanks to my family. This work could not have been possible without the unrelenting support of my family, particularly my husband, Mr. Saurabh Bansal. He has always been willing to extend his helping hand and encouraged me to carry on the work in moments when at times, I got a little disheartened. He supported me in every possible way to ease my research journey. I am really grateful to him for his continued and unfailing love and support. My parents and parents in-law helped me a lot by taking upon themselves the entire responsibility of the household work and giving me enough time to concentrate on this research work. I pay my sincere regards to them for their unconditional love and for inculcating confidence in me to pursue a career in academics. The work would not have been possible without the smiles of my son, Anay Bansal which used to cheer up my day and encourage me to overcome all the difficulties.

Ankita Bansal

ABSTRACT

Prediction of quality attributes to improve software quality is gaining significant importance in the research. A number of metrics measuring important aspects of an object oriented program such as coupling, cohesion, inheritance and polymorphism have been proposed in the literature. Using these metrics, the quality attributes such as maintainability, fault proneness, change proneness, reliability etc. can be predicted during the early phases of the software development life cycle. Various models establishing the relationship between software metrics and quality attributes can be constructed, which can be used by researchers and practitioners in improving the software quality.

Faults and changes in the software are inevitable. This is due to large sized, complex software and presence of inadequate resources (time, money and manpower) to completely test the software. Additionally, there are ongoing changes in the software due to multiple reasons such as change in user requirements, change in technology, competitive pressure etc. Given this scenario, it is very important to predict the changes and faults during the early phases of software development life cycle leading to better quality and maintainable software at a low cost. Identification of change and fault prone parts of the software, help managers to allocate resources more judiciously, thereby leading to reduction of costs associated with software development and maintenance. Testing and inspection activities can be disproportionately focused

on the change and fault prone parts of the design and code.

In literature, there are few prediction models proposed to predict change prone parts of the software. Therefore, a structured review is very important to provide commonalities and differences between the results of these studies. We have formulated various research questions according to which we have compared and reviewed a number of the software change proneness models for object oriented software. The research questions formulated in the review helped in identifying gaps in the current research and future guidelines have been proposed which can be used by software practitioners in future research.

To gather insights into the quality and reliability of the open source software, we have used a number of popular open source software for the purpose of empirical validation. The literature shows that majority of the prediction models are trained using the historical data of the same project. There are broadly two approaches for predictive analysis, machine learning and statistical. Both these approaches are inherently different, raising the question that which approach is better than the other. Besides this, another question that keeps revolving in the minds of researchers is, "Among multiple machine learning techniques available, which classifier should be used for accurate prediction?" To investigate these questions, we have compared the performance of 15 data analysis techniques (14 machine learning and one statistical) on five official versions of the Android operating system. In other words, we have constructed various metric models using the machine learning and statistical techniques.

Literature shows that metric models are widely used for identification of change and fault prone classes. However, training these models using machine learning and statistical techniques is a time consuming task and thus, it is not feasible on a daily basis to use these models. An alternative is to define thresholds of metrics which can be used for predicting change and fault prone parts. Thresholds, also known as risk indicators, define an upper bound on the metric values such that the classes having metric value above thresholds are considered to be potentially problematic. Identifying thresholds helps developers, designers and testers to pay focused and careful attention on these risky (or problematic) classes. We have identified threshold values of various object - oriented metrics of different open source software to predict change and fault proneness. A statistical approach based on logistic regression is used to calculate the threshold values. Another approach to calculate the threshold values is based on receiver operating characteristics curve. We have explored both the approaches to calculate threshold values of the metrics of different software.

There are studies of inter - project validation for fault prediction; however, there is limited research regarding cross-project validation for change prediction. In this research, we have conducted inter - project validation for change prediction using 12 open source datasets obtained from three software. Testing the prediction models on the same data from which they are derived is some what intuitive, hence inter - project validation can help in obtaining generalizable results.

Contents

List of Tables

List of Figures

List of Publications

Papers Accepted/Published in International Journals

1. Malhotra R., Bansal A.: "Fault Prediction Considering Threshold Effects of Object Oriented Metrics", *Expert Systems*, vol. 32, no. 2, pp. 203-219, 2015 (impact factor: 0.761).

2. Malhotra R., Bansal A.: "Predicting Software Change in an Open Source Software using Machine Learning Algorithms", *International Journal of Reliability Quality and Safety Engineering*, vol. 20, no. 6, pp. 1-14, 2014.

3. Malhotra R., Bansal A.: "Quantitative Assessment of Risks Considering Threshold Effects of Object Oriented Metrics using Open Source Software", *Software Quality Professional*, vol. 14, no. 4, pp. 33-46, 2012.

4. Malhotra R., Bansal A.: "Software Change Proneness Prediction: A Literature Review", *International Journal of Computer Applications in Technology*.

5. Malhotra R., Bansal A.: "Prediction of Change Prone Classes using Threshold Methodology", *Advances in Computer Science and Information Technology*, vol. 2, no. 11, pp. 30-35, 2015.

6. Malhotra R., Bansal A.: "Identifying Threshold Values of an Open Source Software using Receiver Operating Characteristics Curve", *Journal of Information*

and Optimization, Taylor and Francis.

Papers Accepted/Published in International Conferences

7. Malhotra R., Bansal A.: "Improving Software Quality by Predicting Change Proneness in an Open Source Software", *International Conference on Optimization Modeling and Applications*, New Delhi, India, 2012.

8. Malhotra R., Bansal A.: "Cross Project Change Prediction Using Open Source Projects", *3rd International Conference on Advances in Computing, Communications and Informatics, IEEE,* New Delhi, India, 2014.

9. Malhotra R., Bansal A.: "Prediction of Change Prone Classes using Machine Learning Methods and Statistical Methods", *International Conference, Software and Emerging Technologies for Education, Culture, Entertainment, and Commerce: New Directions in Multimedia Mobile Computing, Social Networks, Human-Computer Interaction and Communicability*, Venice, Italy, 2012.

10. Malhotra R., Bansal A.:"Predicting Change using Software Metrics : A Review", *4th International Conference on Reliability, Infocom Technologies and Optimization, IEEE*, New Delhi, India, 2015.

Papers Communicated in International Journals/Conferences

11. Malhotra R., Bansal A.: "Analysis of Various Data Analysis Techniques to Identify Change Prone Parts of an Open Source Software: A Replicated Study",*Journal of System and Software, Elsevier*, 2015.

12. Malhotra R., Bansal A.: "Identifying and Validating Threshold Values of Object Oriented Metrics for Change Prediction", *Journal of Information Processing*, 2015.

13. Malhotra R., Bansal A.: “Investigation of Ensemble Learners for Prediction of Change Proneness in an Open Source Software”, *Software Quality Professional, ASQ*, 2015.

Abbreviations

AB	Adaboost
ACC	Average Cyclomatic Complexity
ADT	Alternating Decision Tree
AHF	Attribute Hiding Factor
AID	Access of Imported Data
AIF	Attribute Inheritance Factor
ALD	Access of Local Data
ALOC	Average Number of Lines of Code
ALOCO	Average Number of Lines with Comments
ANFIS	Adaptive Neuro-Fuzzy Inference System
ANN	Artificial Neural Network
AUC	Area Under the ROC Curve
BDM	Behavioral Dependency Measure
BLOC	Number of blank lines
BN	Bayesian Networks
CBO	Coupling between Objects
CD	Critical Distance
CF	Coupling Factor
CFS	Correlation-based Feature Selection
CHAID	Chi-squared Automatic Interaction Detection

CK	**Chidamber and Kemerer**
CMC	**Class Method Complexity**
CMS	**Configuration Management System**
CRT	**Classification and Regression Tree**
CTA	**Coupling Through Abstract data type**
CTM	**Coupling Through Message passing**
CVS	**Concurrent Versioning System**
DAC	**Data Abstraction Coupling**
DCRS	**Defect Collection and Reporting System**
DIT	**Depth of Inheritance Tree**
DPA	**Dynamic Polymorphism in Ancestors**
DPD	**Dynamic Polymorphism in Descendants**
DT	**Decision Tree**
DTNB	**Decision Table Naive Bayes**
EL	**Ensemble Learners**
ELOC	**Number of Executable Lines of Code**
ICP	**Information flow-based Coupling**
IH-ICP	**Information flow-based Inheritance Coupling**
LALO	**Langage d’agents Logiciel Objet**
LB	**LogitBoost**
LCC	**Loose Class Cohesion**
LCOM	**Lack of Cohesion in Methods**
LMT	**Logistic Model Trees**
LOC	**Lines Of Code**
LR	**Logistic Regression**
MHF	**Method Hiding Factor**
MIF	**Method Inheritance Factor**

ML	**Machine Learning**
MLP	**Multi-layer Perceptrons**
MNOB	**Maximum Number Of Branches**
MOOD	**Metrics for Object-Oriented Design**
MPC	**Message Passing Coupling**
NAC	**Number of Ancestor Classes**
NASA	**National Aeronautics and Space Administration**
NB	**Naive Bayes**
NBTree	**Naive Bayes Decision tree**
NDC	**Number of Descendant Classes**
NIC	**Number of Imported Classes**
NIH-ICP	**Non-Inheritance information flow-based Coupling**
NIM	**Number of Instance Methods**
NIV	**Number of Instance Variables**
NLDM	**Number of Local Default Visibility Methods**
NLM	**Number of Local Methods**
NLOC	**Number of uncommented Lines Of Code**
NMIMP	**Number Of Methods Implemented in a class**
NOA	**Number of Attributes**
NOC	**Number of Children**
NOLV	**Number Of Local Variables**
NOM	**Number of Methods per Class**
NumPara	**Number of Parameters of the methods implemented in a class**
NOP	**Number Of Parameters**
NPM	**Number of Public Methods**
NPRM	**Number of Private Methods**
NPROM	**Number of Protected Methods**

OO	**Object-Oriented**
OS	**Operating System**
OSS	**Open Source Software**
OVO	**Overloading in stand-alone classes**
PC	**Principal Component**
PF	**Polymorphism Factor**
PMR	**Performance Management Traffic Recording**
RBL	**Rule Based Learning**
RBF	**Radial Basis Function**
RF	**Random Forest**
RFC	**Response for a Class**
ROC	**Receiver Operating Characteristic Curves**
RQ	**Research Question**
SDLC	**Software Development Life Cycle**
SLOC	**Source Lines Of Code**
SPA	**Static Polymorphism in Ancestors**
SPD	**Static Polymorphism in Descendants**
SVM	**Support Vector Machine**
TCC	**Tight Class Cohesion**
VFI	**Voting Feature Intervals**
VP	**Voted Perceptron**
WMC	**Weighted Methods per Class**
XSLT	**EXtensible Stylesheet Language Transformations**

Chapter 1

Introduction and Literature Survey

1.1 Introduction

Producing high quality software is a major challenge for large and complex sized software systems. Some important attributes of software quality include maintainability, reliability, usability and adaptability. Estimating these attributes of a software system is a promising approach towards software quality prediction and thus improving the quality of the software. The reliability of complex and large sized software needs to be ensured to avoid failures and faults, which may not only have huge financial implications, but also risk to life and property. There are several software metrics proposed in the literature [2, 3, 7, 25, 26, 35, 65, 88, 91] which measure the internal characteristics of software such as size, complexity, coupling, cohesion etc. These metrics can be used to predict the external attributes which determine quality of the software. An important challenge is identifying which metrics are useful predictors of software quality attributes. The use of metrics for effective model construction which can be used to estimate or predict quality is a well-known idea. These models can be used by the organizations during the early stages of software development for prediction of faulty or change prone classes leading to improvement in quality. It allows the developers to pay careful and focused attention on such classes leading to substantial saving of time and resources. There exists several prediction models and more are emerging. However, these models have been constructed using different sets of metrics and evaluated using different software. Thus, more such empirical studies should be conducted to allow fair evaluation of all the studies. Besides this, another technique or approach which can be used for quality improvement is identification of thresholds for various metrics. There is lack of work on exploring the potential uses of thresholds for improving software quality.

The work described in this thesis focuses on exploring various techniques and models for improving software quality. In other words, we have constructed various prediction models by establishing a relationship between Object Oriented (OO) metrics and quality attributes such as change and fault proneness. Besides this, we have also explored various techniques and methodologies for identifying thresholds of various metrics. For the purpose of empirical validation, we have used a number of software systems obtained from the Open Source Software (OSS) repositories. Finally, for validating or testing the models and methodologies used for evaluating software quality, we have used a validation technique known as cross - project validation which has not been used in literature to a much extent. Thus, we have explored a relatively newer technique of validation.

The remainder of the chapter gives brief introduction of the field and explains the basic concepts of the work carried out in this thesis.

1.2 Software Quality Attributes

Whenever we want to judge the effectiveness or capability of software, the quality of software comes to our mind. The software organizations desire to produce a software that is highly maintainable and is of good quality. The definition of the term 'quality' is questionable and understood differently by different organizations and people. According to ANSI Standard (ANSI/ASQC A3/1978) "Quality is the totality of features and characteristics of a product or a service that bears on its ability to satisfy the given needs".

A variety of quality models have been proposed in the literature to describe and manage software quality. One of the explicit aims of various quality models is to make quality measurable. Beginning with hierarchical models, according to Wallmuller [144], one of the oldest and most frequently applied models is that of McCall et al. [109]. McCall quality model attempts to bridge the gap between the users and developers by focusing on a number of software quality factors. McCall quality model was followed by quality model presented by Boehm in the year 1978 [19]. Boehm's quality model improves upon the work of McCall and addresses the contemporary shortcomings of models that automatically and quantitatively evaluate the quality of software. Boehm's quality model like McCall model represents a hierarchical structure of characteristics, each of which contributes to the total quality. The more recent model similar to McCall and Boehm is the model developed by Dromey [42], which is focusing on the relationship between the quality attributes and the sub attributes, as well as attempting to connect software product properties with software quality attributes. Besides these important and known models, many more diverse models have been given in literature. Classification of the diverse models include taxonomic models like the ISO 9126, metric-based models like the maintainability index (MI) [37] and stochastic models like reliability growth models (RGMs) [93]. The difference in the models is due to different purposes they serve: The ISO 9126 is mainly used to define quality (definition model), metric-based approaches are used to assess the quality of a given system (assessment model) and reliability growth models are used to predict quality (prediction model).

Quality has many characteristics or attributes and some are also related to each other. Some of the important quality attributes are listed in the figure 1.1.

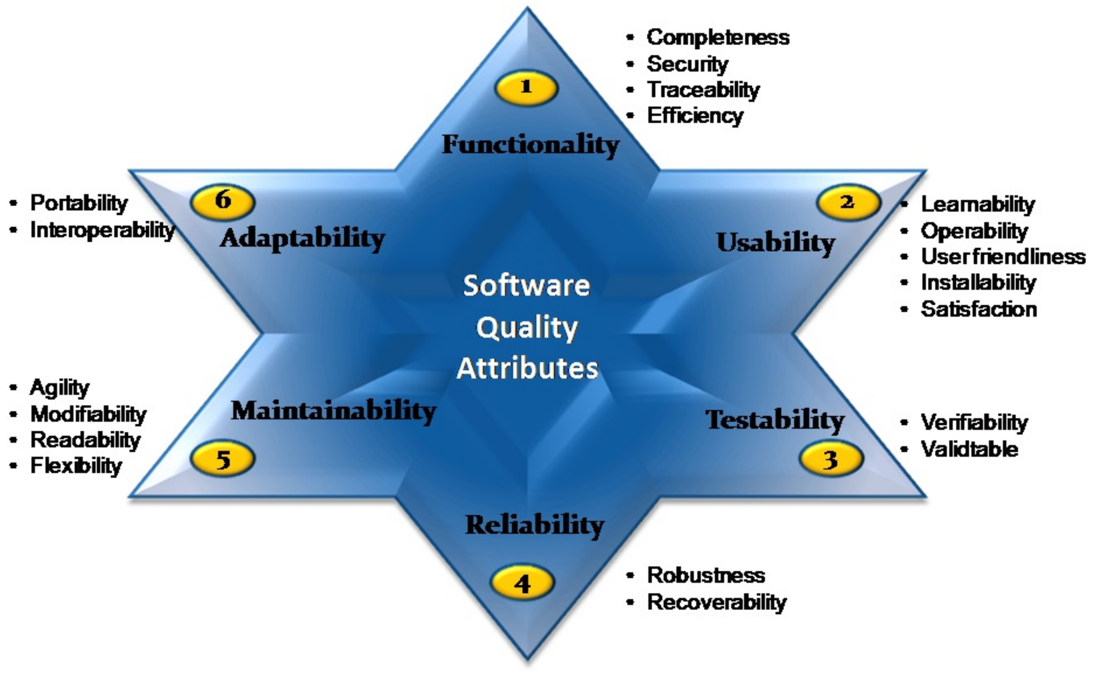


Figure 1.1: Quality Attributes [95]

The brief description of six quality attributes are given below [95]:

- Functionality: The degree to which a software meets its specified purpose.

- Usability: The extent of effort required to learn, operate and understand the functions of the software.
- Testability: The effort required to test a software to ensure that it performs its intended functions.
- Reliability: The extent to which a software performs its intended functions without failure.
- Maintainability: The effort required to locate and fix an error during maintenance phase.
- Adaptability: The extent to which a software is adaptable to new platforms and technologies.

1.3 Software Metrics

From the earliest days of software engineering, software measurement is one of the important aspects of good software engineering practices. With the help of various measurement activities, we get clues of the weak and strong areas in the software development process. Measurement helps one to have a closer and clearer look into the specific characteristics of our processes and products. In other words, measurement activities keeps one actively engaged in all the phases of the software development process. Hence, due to various benefits of software measurement, it is clear that everything should be measurable. If it is not measurable, we should make an effort to make it measurable [4]. Software metrics are widely used for measuring software processes and products. Software metrics can be defined as “The continuous application of measurement based techniques to the software development process and its products to supply meaningful and timely management information, together with the use of those techniques to improve that process and its products” [61]. Various quantitative and qualitative decisions in software projects are done with the help of software metrics. Besides this, with the help of software metrics, we can predict various development resources, track development progress, understand the requirements and maintenance costs etc. [51]. The software metrics can be categorized in a number of ways. For example, software metrics can be categorized as primitive - computed, process - product and subjective - objective [51]. Primitive metrics are those that can be directly observed, e.g. Lines Of Code (LOC), number of defects etc. Whereas, computed metrics are those that cannot be directly observed and must be computed, e.g. metric used to measure quality; i.e. defects/KLOC. Process metrics measure the software development process, such as type of methodology used, experience level of programming staff etc. Whereas, product metrics measure the software products produced at different stages of Software Development Life Cycle (SDLC). For example, product metrics can be used to measure size of final program, complexity etc. Objective metrics result in identical values when measured by more than one observer. For e.g. LOC is an objective metric as any observer would report identical value for it, provided the definition of LOC being used by each observer is consistent. Whereas, subjective metrics may produce different values for a given metric, since their value is judgment dependent. For example, the classification of the software as “organic,” “semi-detached,” or “embedded,” necessary in the COCOMO cost estimation model is dependent on the observer’s judgment [18].

Lots of software metrics have been proposed and used in literature. There are size oriented metrics which are based on the “size” of the software produced. The first and simplest measure of size has been the number of LOC. But LOC measure has few disadvantages such as it is language and programmer dependent. Also, it is not clear whether the comment lines, data declarations, blank lines etc. should be counted in this metric or not. Another size measure used was proposed by Halstead, which is named as Science Measures [65]. Besides this, size can be measured in terms of function points as proposed by Albrecht [7]. Functions points measure the functionality from users point of view. Besides this, there are various testing metrics, complexity metrics, the metrics to measure cohesion and coupling etc. One of the early measures of cohesion and coupling was in the form of Information flow metrics [75]. The multiplication of two terms, fan-in and fan-out was used to define the information flow metrics. Most widely used complexity measure was proposed by McCabe [108]. Zuse [156] provided the study of 18 different categories of software complexity metrics. These earlier metrics (Halstead, McCabe, LOCs) focused at traditional procedural-oriented languages.

Nowadays, OO paradigm is gaining importance. The various OO concepts include classes, object, attributes and methods. Due to number of properties of OO paradigm such as inheritance, encapsulation, abstraction, data hiding etc., it is popularly being used for software development. To measure various dimensions of OO paradigm, a number of OO metrics [2, 25, 26, 35, 88, 91] have also been proposed in the literature in the last two decades. A number of the important OO metrics proposed in the literature are briefly described in section 1.5.1.

1.4 Software Quality Prediction

Software quality prediction involves the prediction of various software quality attributes in the early phases of software development life cycle to improve the overall quality of a software product. This can be carried out by constructing empirical models which can be used by researchers and practitioners to judge or estimate the quality of a software product. These empirical models allow one to predict the external quality attributes as a function of various measurable internal attributes. These external quality attributes are referred as target attributes (dependent variable) and the internal attributes are known as predictor attributes (independent variable). The construction of empirical models takes place in three consecutive phases: training, testing and predicting the unknown data. The process of construction is shown in figure 1.2.

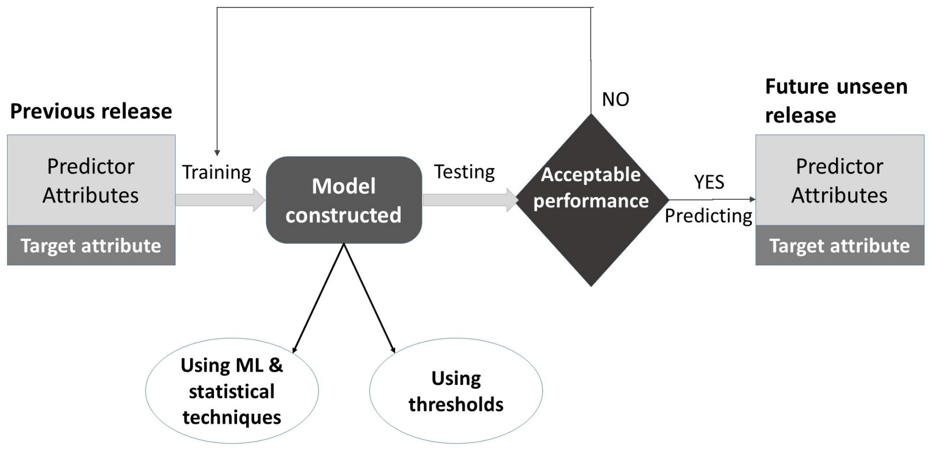
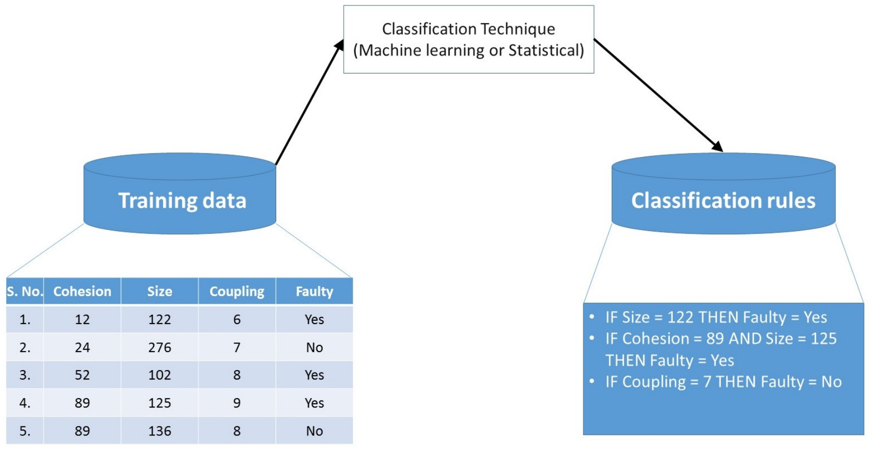


Figure 1.2: Quality Prediction Process

The first phase, i.e. the training or the learning phase uses the training data (for which the values of the target attribute are known) for building a model. Different classification techniques are applied on a part of the training data to build the model by establishing a relationship between the predictor attribute values and the target attribute values. These relationships are captured in the model in form of various classification rules which can be used in the next two phases, i.e. for testing as well for prediction. This process of training the model is also demonstrated in figure 1.3.



S. No.	Cohesion	Size	Coupling	Faulty
1.	12	122	6	Yes
2.	24	276	7	No
3.	52	102	8	Yes
4.	89	125	9	Yes
5.	89	136	8	No

- IF Size = 122 THEN Faulty = Yes
- IF Cohesion = 89 AND Size = 125 THEN Faulty = Yes
- IF Coupling = 7 THEN Faulty = No

Figure 1.3: Training Process

The next phase is known as the testing phase. The constructed classification model or the classification rules can be applied on the remaining part of the training data (which was not used for training) with the known target values to obtain the value of the target attribute known as the predicted value. Such data is also known as test or evaluation data. The predicted value which is obtained from the model is compared with known value of the target attribute to measure the predictive capability or accuracy of the model. If the accuracy of the model is considered acceptable, then it can be used on the new data with unknown values of the target variable. This step is known as the prediction of the unknown data or the future unseen data. Using the classification rules, the value of target variable can be predicted.

In this research, model construction is carried out using the following two approaches:
1. Developing various metric models with the help of different machine learning and statistical methods.
2. Identifying thresholds of various metrics which can be used as quality benchmarks to assess and compare products. Threshold provides an upper bound on the metric values such that the classes having metric values above the threshold value are considered to be problematic, whereas the values lower are considered to be acceptable.

1.5 Literature Survey

In this section, we first provide the current state of research on OO metrics. We then summarize the empirical studies related to the construction of quality prediction models using Machine Learning (ML) and statistical techniques, followed by the empirical studies which identify the thresholds for quality assessment.

1.5.1 Object Oriented Metrics

The first and most popular metric suite was proposed by Chidamber and Kemerer (CK) in 1991 [35]. Another paper by Chidamber and Kemerer defined and validated metrics suite for OO design [36]. The metric suite consists of six metrics namely Coupling Between Object classes (CBO), Response For a Class (RFC), Weighted Methods per Class (WMC), Depth of Inheritance Tree (DIT), Number Of Children (NOC), and Lack of Cohesion in Methods (LCOM). Li and Henry collected the maintenance data in terms of number of lines changed per class to validate the metrics they proposed- Message Passing Coupling (MPC), Data Abstraction Coupling (DAC), Number of Methods (NOM), SIZE1, and SIZE2 [88]. Li proposed a metrics suite consisting of six metrics - Number of Ancestor Classes (NAC), Number of Local Methods (NLM), Class Method Complexity (CMC), Number of Descendent Classes (NDC), Coupling Through Abstract data type (CTA) and Coupling Through Message passing (CTM) [87]. In 1996, Henderson-sellers [74] redefined the LCOM metric given by [36]. Bieman and Kang [15] proposed two metrics to measure cohesion-Tight Class Cohesion (TCC) and Loose Class Cohesion (LCC). Lee et al. proposed coupling metrics which differentiated between inheritance-based and non inheritance-based coupling- Non Inheritance information flow-based coupling (NIH-ICP), Information flow-based inheritance coupling (IH-ICP) [85]. The sum of NIH-ICP and IH-ICP metrics led to Information flow-based coupling (ICP) metric. Abreu proposed another metrics suite consisting of six system level metrics known as MOOD (Metrics for Object-Oriented Design)- Method Hiding Factor (MHF), Attribute Hiding Factor (AHF), Method Inheritance Factor (MIF), Attribute Inheritance Factor (AIF), Polymorphism Factor (PF), and Coupling Factor (CF) [3]. Lorenz and Kidd proposed a set of metrics grouped into four categories size, inheritance, internals, and externals [91]. Size oriented metrics focused on the number of attributes and operations of an individual class, inheritance based metrics concentrated on the way in which operations were reused through class hierarchy, metrics for class internals examined cohesion, while external metrics examined coupling and reuse [91]. Briand et al. [27] proposed a number of coupling metrics which cover different concepts of C++ language: friendship, classes, specialization, and aggregation. An unified framework for measuring cohesion and coupling was given by Briand et al. [27, 25]. Benlarbi and Melo defined a suite of five polymorphism metrics in 1999 - Overloading in stand-alone classes (OVO), Static Polymorphism in Ancestors (SPA), Static Polymorphism in Descendants (SPD), Dynamic Polymorphism in Ancestors (DPA), and Dynamic Polymorphism in Descendants (DPD) [14]. These metrics measure both types of polymorphism, i.e. run time polymorphism and compile time polymorphism.

1.5.2 Software Quality Prediction Models Using Statistical or Machine Learning Techniques

Empirical research involving the development of software quality prediction models using various ML and statistical techniques has been conducted to a large extent. In this research, one of the main contribution is the construction of such prediction models for various OO software which can be used to assess quality of software in earlier phases of the software life cycle. These models use various structural measures for predicting external quality attributes. There are various structural measures or metrics which are found to be significant predictors of quality attributes. Hence, such metrics can be extensively used to construct models which can be used to improve quality.

The empirical study by Lindvall studied the effect of size of a class on the probability of change in that class [89]. The results concluded that the changed classes were generally large in size while unchanged classes were generally small. Koru and Tian [81, 82] worked on Pareto's law, 80:20 principle which states that large majority (around 80%) of problems are rooted in a small proportion (around 20%) of the modules. They proposed a method of ranking to find top-change modules and those having top measurement values. They also used a voting mechanism to identify the modules with the largest size, highest coupling, highest cohesion and largest inheritance. The authors concluded that when ranking or voting mechansim is used, top change modules were not the top-measurement modules. Han et al. developed design models for predicting change proneness [66]. They calculated Behavioral Dependency Measure (BDM) for predicting change proneness and the results concluded that BDM is as an effective indicator for change proneness prediction. Along with BDM, the effect of polymorphism and inheritance relationships has also been taken into account by their another study to predict change proneness [67]. The results concluded that BDM is a significant predictor when system contains high degree of inheritance and polymorphism. Zhou et al. investigated the confounding effect of class size on the relationship between OO metrics and change proneness [153]. The class size was measured using three size metrics- Source Lines Of Code (SLOC), Number Of Methods Implemented in a class (NMIMP) and sum of the Number of Parameters of the methods implemented in a class (NumPara). The analysis of the confounding effect of size was based on linear regression equations. The authors concluded that there is confounding effect of size on the relationship between the metrics and change proneness and thus, should be removed to avoid misleading results. A recent study by Lu et al. found significant relationship between OO metrics and change proneness [92]. The study considered a large number of OO metrics (62) covering four dimensions (size, cohesion, coupling and inheritance) and used 102 Java systems to analyze the change-proneness predictive ability of these OO metrics. To compute the average change proneness predictive ability of OO metrics over all the systems, random effect meta-analysis models were used. The results concluded that the size metrics have moderate ability to identify change and non - change prone classes, coupling and cohesion metrics have lower predictive ability than size metrics and inheritance metrics have poor predictive ability. The study by Malhotra and Khanna [104] determined the effectiveness of ML techniques for predicting change prone classes and compared their performance with the statistical technique. The authors concluded that the performance of ML techniques is comparable to the statistical technique. In addition to metrics, design patterns also play a key role in identifying change and not change prone classes. Some studies [16, 17, 119, 120] have also been done to find effect of design patterns on change proneness. Bieman et al. examined five evolving systems to analyze the relation between design patterns and change proneness [17]. In four of the five patterns, classes were found to be less change prone while in one system pattern, classes were more change prone. Thus, the authors concluded that there is significant relationship between design patterns, other design attributes (class size, inheritance) and changes. The authors Posnett et al.also identified the influence of patterns and metapatterns on change proneness [120]. They also studied the effect of size and concluded that size is a stronger determinant of change proneness than either design pattern or metapatterns.

In addition to the change prediction models, there are measures or metrics which have been used in literature to predict fault prone classes and have been concluded as the significant predictors of fault proneness. Basili et al. [10] worked on eight medium - sized systems developed by the students and found that except LCOM all of the remaining CK metrics [36] are associated with fault proneness. Tang et al. [136] worked on three industrial projects developed in C++ and concluded that among CK metrics, WMC and RFC are effective predictors of fault proneness. Cartwright & Sheppard [30] worked on OO constructs such as inheritance and polymorphism and concluded that the classes using high inheritance have more defect densities. Emam et al. [48] analyzed and used inheritance metrics (NOC and DIT) and various coupling metrics to predict fault prone classes. They concluded that DIT and export coupling measures are strongly associated with fault-proneness. The same result was reported by Glasberg et al. [59]. Confounding effect of size is taken into account by various studies and they concluded that class size is associated with many contemporary OO metrics [27, 30, 47, 63]. Thus, the results should be revalidated after considering the effect caused by metrics. Briand et al. have empirically validated a set of 49 metrics to construct suitable fault proneness models [22, 24]. They used univariate and multivariate analysis to find the individual and combined effect of metrics on fault proneness. The authors in the study [22] worked on eight systems and found all the metrics to be significant except NOC. Whereas, another study by Briand et al. [24] used a commercial system and found WMC, RFC and CBO to be the strong predictors. They also found DIT metric to be related to fault proneness but in an inverse manner. Another study by Briand et al. [28] used two commercial systems to find the relationship between polymorphism, CK and some OO metrics with fault proneness. They constructed models for predicting fault prone classes using two regression techniques, multivariate adaptive regression splines and Logistic Regression (LR). The results showed that the model predicted using multivariate adaptive regression splines outperformed the LR model. Yu et al. [149] chose a set of eight internal product metrics (measuring important design attributes of software such as size of software, coupling, cohesion, inheritance and reuse) and examined the relationship between these metrics and fault proneness. They conducted their study using a large network service management system developed by three professional software engineers and found some of the metrics (CBOin, RFCin, and DIT) to be insignificant while the other metrics to be significant predictors of fault proneness. Some authors [53, 63, 79, 118] constructed various ML models to predict fault prone classes. Gyimothy et al. [63] used regression (linear and logistic) and ML techniques (neural network and decision tree) to construct fault prediction models. Kanmani et al. [79] introduced neural network based prediction models whereas Fenton et al. [53] and Pai et al. [118] introduced bayesian belief networks to predict faulty classes. Few papers [130, 134, 152] discussed and taken into account the severity of faults to predict faulty classes. Zhou and Leung [152] validated NASA (National Aeronautics and Space Administration) software to predict fault proneness models with respect to two categories of faults: high and low. Singh et al. [134] worked on the same software to predict fault proneness models but have categorized faults with respect to all the severity levels given in NASA software. This allows to distinguish between different types of failures and thus, provides more meaningful, correct and detailed analysis of fault data.

As discussed in this section, there are various studies which have validated metrics to predict external quality attributes, fault proneness and change proneness. However, most of these studies are based on intra- project validation, i.e. training the model using the data of the historical releases and applying them to predict fault prone classes of the upcoming or future release of the same project. There are few studies which have focused on cross-project predictions for fault proneness, i.e. training and testing the model using different projects. However, some of the studies [114, 143, 155] which carried out cross-project predictions concluded that cross-project prediction is a big challenge and models are more accurate when they are trained using the data of a same or similar project. Watanabe et al. [145] and Whayudin et al. [143] commented that data characteristics are important for cross-project defect prediction and due to different characteristics of each project; cross-project defect prediction is a big challenge. Watanabe et al. [145] trained a prediction model from a Java project and applied it to a C++ project. When carrying out inter-project predictions, the authors proposed a method known as 'metrics compensation' which should be used to compensate the test data. They observed that both recall and precision increased after using 'metrics compensation' method for model adapting. Zimmermann et al. [155] ran 622 cross - project defect predictions on 12 real world applications and found that cross-project prediction is a serious challenge. The study found that only 21 predictions worked successfully (all precision, recall, and accuracy are greater than 75%). They also concluded that cross-project predictions are not symmetrical. This means if project A can predict the defects of project B well, this doesn't imply the reverse is also true. The authors Turhan et al. [140, 141] also conducted cross-company defect prediction on 10 projects from 8 different companies. They concluded that cross-company predictions increase the probability of defect detection at the cost of increasing false positive rate. He et al. [73] investigated cross-project predictions on 34 public datasets obtained from 10 software. They used five ML techniques to construct prediction models. Their study focused on the selection of suitable training data for the projects without historical data. They proposed an approach based on the distributional characteristics of the software to select suitable training data. They concluded that the prediction results obtained by using the training data obtained using their proposed approach are comparable with those provided by training data from the same project.

1.5.3 Software Quality Assessment Using Thresholds of Software Metrics

As can be seen from the previous section, there are number of empirical studies which have established relationship between software metrics and external quality attributes. However, these studies have not exploited the usage of thresholds to predict the external attributes. In this section, we discuss the empirical studies which have identified and calculated the thresholds of various metrics. There is lack of quantitative threshold models for predicting different quality attributes except fault proneness.

A threshold value of a metric can be defined as an extreme value above which a class is considered to be risky and thus needs focused attention. Some researchers proposed threshold values based on experience and intuition [37, 108, 116]. However, the threshold values based on intuition cannot be universally accepted and generalized. For example, McCabe's complexity metric is given a threshold value of 10 [108]. Similarly, for the maintainability index metric, 65 and 85 are defined as thresholds [37]. Rosenberg [125] suggested threshold values for CK metrics which can be used to select classes for redesign. Henderson-Sellers [74] suggested that the classes can be clustered into three categories- safe, flag, and alarm. These categories represent different levels of program complexity.Thresholds are also defined using mean (μ) and standard deviation (σ). For example, in the study by Erni et al. [49], the minimum and maximum values of threshold (T_{max} and T_{min}) as μ + σ and μ - σ respectively. However, this methodology did not became popular because it can be only used when the metrics are normally distributed which is not the case. French [55] used Chebyshev's inequality theorem (not restricted to normal distribution) in addition to mean (μ) and standard deviation (σ) to derive threshold values. French defined the threshold as T= μ + k * σ (k=number of standard deviations). Although French used Chebyshev's inequality theorem which is not only for normally distributed metrics, however the methodology was confined to only two-tailed symmetric distributions.

Some of the authors have worked on levels of inheritance depth and how they affect maintenance [29, 39, 69, 121]. Daly et al. [39] studied the effect of inheritance depth on effort required to make a maintenance change. They compared the performance of the program with 3 and 5 levels of depth. They concluded that program with 3 levels of depth took less time than the program with no inheritance, whereas, programs with 5 levels took more time. On the other hand, Cartwright [29] concluded that making changes in a program with 3 levels of depth require more effort than the program with no inheritance. In other studies, Harrison et al. [69] and Prechelt et al. [121] also showed similar result as that of the Cartwright's study.

Ulm [142] proposed a statistical technique based on LR for assessing thresholds in epidemiological studies. This statistical technique was used by Benlarbi et al.[13] and El Emam et al. [46] to calculate the threshold values of various metrics. The authors have constructed two models: (1) using the thresholds and (2) without using the thresholds. They concluded that there was no difference in the performance of the two types of models. Another study in the epidemiological field conducted by Bender [12] showed deficiencies in the model proposed by Ulm [142]. The model assumed that the fault probability increases according to the logistic function if the metric value is above the threshold and when the metric value is below the threshold, the probability of fault is constant. Thus, Bender [12] redefined the threshold effects as an acceptable risk level. Shatnawi [129] used the technique proposed by Bender[12] to find the threshold values for CK metrics [36]. They identified a risk level for any arbitrary threshold value. Their results concluded that CK metrics have threshold effects at various risk levels. Besides this, Shatnawi and Li [131] also investigated the use of Receiver Operative Characteristics (ROC) curve to identify threshold values. They tested the threshold values in two categorizations, the binary category (Error and No-error) and the ordinal category. Threshold values were identified to differentiate high-risk error-proneness classes in the ordinal categorization from the No-error classes. Whereas, to differentiate between the two binary categories, the authors could not identify the threshold values.

1.6 Significant Insights

The significant insights and gaps in literature are stated in this section. The gaps identified are stated underneath on which we propose to work on:

1. Lack of Risk Indicators: Thresholds identify certain alarming values of the metrics which are used to find classes that fall outside the acceptable risk level. Thus, they are also known as risk indicators. There is lack of work on exploring the potential usage of thresholds to predict fault prone classes. Few studies have identified the threshold values of only the popularly used CK metrics [36] to predict fault prone classes. For other OO metrics, to the best of our knowledge, there is no study related to our field which has proposed the threshold values. Besides this, to predict to predict change prone classes, no risk indicators or thresholds have been proposed. Thus, due to limited amount of studies, the threshold values cannot be generalized and used across different organizations

which can use them as benchmarks for their projects.

2. Lack of Validation Techniques: The studies in literature used inter - release validation to validate the models, i.e. using the different releases of the same software to train as well as validate. But inter-release validation is not always possible as the historical data (previous release) may not be available, may not exist or the change/fault data may not be available. There are few studies which have done intra - project validation for fault prediction, but to the best of our knowledge, no study has used intra - project validation for change prediction. Thus, the use of intra - project validation should be explored and widely used. Intra - project validation provides generalized results leading to well - formed and acceptable theories.

3. Lack of Statistical Tests: Once the performance of the models are evaluated using various measures, it is very important to statistically evaluate the results to decide whether the differences between the performances of the models are not random or by chance. In other words, it might be possible that without the use of statistical tests, the good performance of a classifier is achieved by-chance. Despite the importance of statistical tests, there is lack of statistical and post hoc tests in the studies related to change and fault prediction.

The basic premise of our work is to address change and fault prediction using OO metrics. Keeping in view the identified gaps, we have discussed two methodologies to predict quality attributes: (1) construction of metric models using ML and statistical techniques and (2) application of threshold methodology. To make our results more generalizable and applicable to a number of domains, we perform cross - project predictions where the training and the validation data belong to two different projects rather than the same project. Further, the models constructed are also statistically evaluated using various statistical and post hoc tests.

1.7 Goals of the Thesis

A number of factors such as increasing size and complexity, inadequate resources, evolving customer requirements and change in the external environment influence the quality of software. Due to inadequate resources and increasing size of the software, large volume of faults is introduced at every phase of SDLC. At the same time, due to the faults in the maintenance phase, change in the customer's requirements and external environment such as technology, there are lots of changes that need to be introduced in software after it is delivered to the customer. Hence, faults and changes in a software are inevitable. Thus, among a number of external quality attributes, we focus on mainly fault proneness and change proneness. In this research, we have performed a comparative review of existing studies and conducted various replicated studies to predict fault and change prone classes using OO metrics. This allowed to draw generalized conclusions and present commonalities and differences across studies. We have analyzed and assessed the performance of various ML techniques using varied software, validation methods and multiple performance measures. Further, we have analyzed the significance of the results using statistical tests. We have also tested and verified the performance of the predicted models using the datasets obtained from different OSS. Besides this, we have explored the usage of thresholds of metrics for prediction of change and fault prone classes. Thresholds, also known an risk indicators represent certain alarming values which are used to find classes that fall outside the acceptable risk level. One of the major contributions of this work is the identification of such risk indicators for various OO metrics used. Another major contribution is the usage of inter - project (also known as cross - project) validation for validating the predicted models. The use of inter - project validation also leads to more generalized results which can be used across organizations. The summary of the goals of the work is provided below:

1. To identify useful and efficient predictors of the quality attributes by establishing an association between internal and external attributes.
   More studies using feature reduction techniques should be conducted which helps to select a subset of predictors which are significant in predicting the dependent variable. These significant predictors can be used by researchers and practitioners for construction of prediction models.

2. To identify **risk indicators which can be used as quality benchmarks** in order to assess and compare products.
   Risk indicators also known as thresholds, provide an **upper bound** on the metric values such that the classes having metric values above the threshold value are considered to be problematic, whereas the values lower are considered to be acceptable.

3. To **carry out inter-project validation** in order to obtain generalized results.

4. To **explore the use of widely known and popular OSS** for change and fault prediction.

There are a number of advantages of OSS which motivated us to use them for prediction such as they are cost effective, they are easy to customize and upgrade and thus, they improve faster and also they are no copyright issues or concerns.

5. To **develop and compare various quality prediction** models using learning techniques.

1.8 Organization of the Thesis

The organization of the thesis is presented in this section. Chapter 1 focuses on the basic concepts and provides a review of the empirical studies. Chapter 2 concentrates on the research methodology followed in this work. Chapter 3 reviews the existing studies and provide future guidelines. The subsequent chapters 4 and 5 focus on constructing prediction models using various ML techniques. Chapters 6, 7 and 8 identifies the risk indicators of various metrics for predicting change and fault prone parts. Chapter 9 conducts cross - project predictions and finally the work is concluded in chapter 10. The brief description about each chapter is as follows:

Chapter 1. This chapter introduces the work in the thesis stating the basic terminology and concepts. The literature review describing the past studies is also included in this chapter.

Chapter 2. This chapter presents the research methodology followed in this thesis. The chapter defines the research problem and summarizes the independent and the dependent variables used. The description of data collection procedure and the software used in this thesis is presented. We also present the details of data analysis techniques (ML and statistical) and the pre - processing steps. The validation techniques, performance measures and statistical tests used in this work are summarized in this chapter.

Chapter 3. In this chapter, we perform a systematic review of the existing literature for prediction of change prone portions of the software. We present Research Questions (RQs) that address various issues related to the software, measures and techniques used in the existing literature for change prediction models. Further, current trends are extracted and future insights are drawn from these studies.

Chapter 4. This chapter compares and analyzes various ML techniques (Adaboost (AB), LogitBoost (LB), Random Forest (RF), J48, BayesNet (BN), NaiveBayes (NB), Bagging, Multilayer Perceptron (MLP)) and compares their performance with the statistical (LR) technique for prediction of change prone classes using OO metrics. The results of this work are validated using 'Java TreeView', an OSS. A subset of metrics are also extracted in this work. The predictive capability of the predicted models is analyzed using the ROC curve.

Chapter 5. In this chapter, we develop change prediction models using a widely used and popular mobile operating system, Android. The main aim of this work is to perform inter - release validation i.e. constructing the models on one release and validating these models on the subsequent releases. For example, model is constructed using Android 2.0 and validated on Android 2.1. The superiority of one technique over the other technique is also computed in this work using statistical test. Futher, we carry out post - hoc analysis to determine which technique is statistically significant over the other technique.

Chapter 6. This chapter determines various risk indicators of OO metrics. These thresholds can be used as alarming values for reducing the occurrence of faulty classes. In this work, we calculate the threshold values using a statistical approach based on the LR methodology. The threshold values computed for the metrics of three software (KC1, Ivy and JEdit) are validated on three other software (Ant, Tomcat and Sakura) using ML techniques.

Chapter 7. In this chapter also, we determine the risk indicators in the form of threshold values of various OO metrics. We have used these indicators for identifying the classes which are prone to change. We used the same LR methodology as is used in chapter 6. The thresholds are computed using different releases of two OSS; Freemind and Xerces. These threshold values are validated on different releases as well as on different software (Frinika and Xalan).

Chapter 8. In this chapter, methodology based on ROC curve is used for identifying the alarming values (thresholds) of various OO metrics. We compare the ROC approach with the approach proposed in chapter 7 that uses LR methodology for computing the threshold values of OO metrics for predicting change prone classes. We conclude that ROC approach is more effective than the LR methodology for prediction of risky classes.

Chapter 9. In this chapter, an extensive validation of developed change prediction models is carried out using different projects. The basic purpose is to identify whether the distributional characteristics of the training set can be used for identifying a different similar nature software on which the models predicted can be applied for the purpose of predicting change prone classes.

Chapter 10. The final chapter includes the conclusions of the research work and suggests a few directions for further research.

Chapter 2

Research Methodology

2.1 Introduction

In order to carry out either an empirical study or to perform a survey of existing empirical studies, it is very important to follow a research methodology. A proper research methodology helps to organize our research in a systematic and planned manner. It allows to understand and define the problem, identify the goals of the study and develop various RQs. In this chapter, an overview of the research process and methodology followed in this thesis is presented. The chapter is organized as follows: Section 2.2 presents the research process followed in performing empirical studies in subsequent chapters. Section 2.3 formulates the research problem. The literature survey conducted in order to provide an overview of the existing studies is presented in section 2.4. In section 2.5, the independent and dependent variables are defined. Section 2.6 describes various data analysis methods used in this research. Section 2.7 presents the empirical data collection. Section 2.8 describes various techniques used to pre-process and analyze the data. The procedure used for constructing of prediction models is explained in section 2.9. Section 2.10 presents various validation methods. Section 2.11 focuses on various model evaluation measures. Finally, various statistical tests used are explained in section 2.12.

2.2 Research Process

Research process defines steps necessary to effectively carry out a research. Figure 2.1 illustrates the research process followed in the chapters presented in this thesis. Each of the steps in this research process is explained in the subsequent sections.

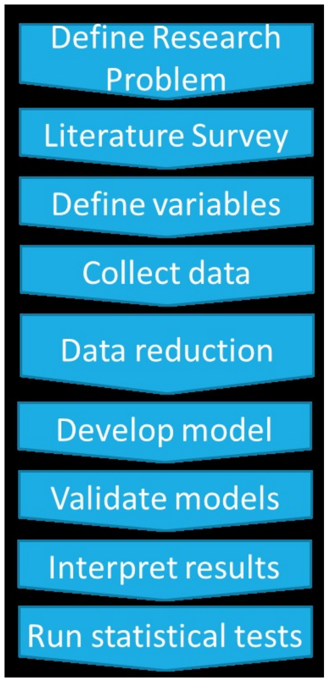


Figure 2.1: Research Methodology

2.3 Define Research Problem

The first and most essential step in a research is to formulate a research problem. When we begin a research, various questions related to the concerned topic start cropping in our minds. We try to search the literature in order to obtain the answers to these questions. The questions that remain unanswered are formulated and defined in the form of RQs. As we progress through the research, we try to search the answers to these questions. The following RQs are addressed in this work:

1. What is the current state of research on predicting change and fault proneness for improving quality of software?
2. What is the qualitative performance of different machine learning techniques in modeling the change and the fault data?
3. What techniques and methodologies can be used for detecting risk indicators?
4. What is the performance of model prediction using open source and widely used software?
5. What is the effective way of validating the prediction models once they are trained?

2.4 Literature survey

There have been various OO metrics [6, 15, 25, 26, 30, 36, 85, 88, 91] proposed in the literature to assess different software quality attributes such as change proneness, fault proneness, maintenance effort of the class, etc. There have been multiple studies to evaluate the impact of OO metrics on the software quality [10, 22, 24, 30, 59, 66, 67, 81, 82, 89, 136, 153] using the statistical and ML techniques. Most of these studies have constructed metric models to judge or predict the quality of the system. Besides this, literature also shows that the quality of the system can also be improved by identifying threshold values (risk indicators) of various metrics. Bender [12] defined thresholds as “Breakpoints that are used to identify the acceptable risk in classes”. Some of the researchers have also used threshold values of metrics for predicting fault prone classes [29, 39, 46, 69, 121, 129].

2.5 Define Variables

In order to develop various software quality models, we need to identify and define the variables. The variables are of two types, dependent and independent variables. The variable that describes the quality attribute that is to be predicted or tested is called the dependent variable (also known as target or predictor variable), whereas, the independent variable (also known as explanatory variable) is a variable that can be changed or varied to see its affect on the dependent variable. In other words, whenever we change the value of the independent variable, the corresponding affect on the dependent variable can be seen. The dependent variables used in this thesis are change proneness and fault proneness. The independent variables used are various OO metrics proposed in the literature.

2.5.1 Independent Variables

Since the OO paradigm is gaining popularity, various OO metrics have been used as the independent variables in the literature. There are various dimensions used in OO design such as coupling, cohesion, inheritance and size that can be measured. Coupling shows the degree of interdependence between the classes. If two classes depend highly on each other, we say they are tightly coupled and this is undesirable. Cohesion on the other hand, measures the interdependence between the variables and methods within a class. One of the most important goals of OO design is to have high cohesion and loose coupling between the classes. Inheritance allows one class to inherit the properties and features of another class. Thus, inheritance metrics deal with the information about ancestors and descendants of a class [5]. A single metric alone is not sufficient to explain all characteristics of software under development. Several metrics must be used together to have better assessment of the software. Among a number of metrics proposed, CK metrics [36] has been widely used in the literature. In this study, we have used various OO metrics including the popularly used CK metrics as the independent variables. The metrics we have used, along with their source and dimension to which they belong are summarized in table 2.1. Following have been the two main reasons behind our decision for selecting the metrics: (1) they measure the important characteristics of the software like size, coupling, cohesion and inheritance and can be easily measured with the help of an open source tool 'Understand for Java' (http://www.scitools.com/features/metrics.php); (2) they are widely accepted by the software engineering community.

Table 2.1: Independent Variables

S.No	Metrics	Definition	Source	Dimension
1.	Lack of Cohesion amongst Methods (LCOM)	For each data field in a class, the percentage of the methods in the class using that data field; the percentages are averaged and then subtracted from 100%.	[36]	Cohesion
2.	Coupling Between Objects (CBO)	Number of classes whose attributes is used by the given class plus those that use the attributes of the given class.	[36]	Coupling
3.	Response For a Class (RFC)	Number of methods in the class including the methods that are called by class's methods.	[36]	Coupling
4.	Depth of Inheritance Tree (DIT)	Maximum number of steps from the class node to the root of the tree.	[36]	Inheritance
5.	Number of Children (NOC)	Number of direct children of a class in a hierarchy.	[36]	Inheritance
6.	Source Lines of Code (SLOC)	Number of lines containing only the source code.	`http://www.scitools.com/features/metrics.php`	Size
7.	Number of Blank Lines (BLOC)	Number of blank lines of code.	`http://www.scitools.com/features/metrics.php`	Size
8.	Number of Executable Lines of Code (ELOC)	Number of lines containing executable source code.	`http://www.scitools.com/features/metrics.php`	Size
9.	Average Number of Lines of Code (ALOC)	Average number of lines containing source code for all nested functions or methods, including inactive regions.	`http://www.scitools.com/features/metrics.php`	Size
10.	Average Number of Lines with Comments (ALOCO)	Average number of lines containing comment for all nested functions or methods, including inactive regions.	`http://www.scitools.com/features/metrics.php`	Size
11.	Average Cyclomatic Complexity (ACC)	Average cyclomatic complexity for all nested functions or methods.	`http://www.scitools.com/features/metrics.php`	Size

S.No	Metrics	Definition	Source	Dimension
12.	Number of Instance Methods (NIM)	Counts the number of instance methods in a class.	[91]	Size
13.	Number of Instance Variables (NIV)	Counts the number of instance variables.	[91]	Size
14.	Number of Local Methods (NLM)	Counts the number of local (not inherited) methods.	`http://www.scitools.com/features/metrics.php`	Size
15.	Number of Attributes (NOA)	Counts the number of attributes/variables defined in a class.	`http://www.scitools.com/features/metrics.php`	Size
16.	Number of Methods per Class (NOM)	Counts the number of methods defined in a class.	`http://www.scitools.com/features/metrics.php`	Size
17.	Number of Public Methods (NPM)	Counts the number of local (not inherited) public methods.	`http://www.scitools.com/features/metrics.php`	Size
18.	Number of Private Methods (NPRM)	Counts the number of local (not inherited) private methods.	`http://www.scitools.com/features/metrics.php`	Size
19.	Number of Protected Methods (NPROM)	Counts the number of local protected methods.	`http://www.scitools.com/features/metrics.php`	Size
20.	Number of Local Default Visibility Methods(NLDM)	Counts the number of local default visibility methods.	`http://www.scitools.com/features/metrics.php`	Size
21.	Weighted Methods Per Class (WMC)	Count of sum of complexities of all methods in a class.	[36]	Size

2.5.2 Dependent Variable

In this work, we have used two binary dependent variables, i.e. change proneness and fault proneness.

- Changes in software are inevitable due to various reasons such as change in user requirements, up-gradation in technology etc. Therefore, forecasting the classes that are change prone in the future or upcoming release of the software would be highly beneficial. Thus, we have used dependent variable as change proneness and investigated the relationship between change proneness and various OO metrics. Change proneness is defined as the probability of prediction of change in the successive releases of the software. This prediction is based on the changes in the current or present release. We have compared the classes of the two releases and measured changes in terms of number of lines of code added, deleted or modified in the class of first (previous) release with respect to the class of the same name in the current release.
- Another dependent variable used in this study is fault proneness. Fault proneness is defined as the probability of occurrence of fault in a class [5].We have used various classification and regression methods which are based on event probabilities to predict fault proneness. An event is a detection of fault in a class during the SDLC.

2.6 Selection of Data Analysis Methods

In this work, we have predicted various models using statistical and ML techniques. We have used a popular statistical method, i.e. LR to predict various quality models. In addition to statistical method, we have also used various ML techniques which can be categorized under the following types: Decision Trees (DT), Bayesian Networks (BN), Ensemble Learners (EL), Artificial Neural Networks (ANN) and Support Vector Machines (SVM). In this section, we would briefly explain each of these ML and statistical methods.

2.6.1 Logistic Regression

LR is one of the commonly used statistical modeling method. LR is used to predict the dependent variable from a set of independent variables (a detailed description is given by [5, 10, 76]. There are two types of LR [76]: (a) Univariate LR; (b) Multivariate LR. We have used both univariate and multivariate LR in this work to produce the relevant results. Univariate LR finds the association between each individual independent variable (OO metrics) and the dependent variable (change proneness and fault proneness). Multivariate LR is used to construct the prediction models for predicting change and fault prone classes. The combination of all the OO metrics is considered for prediction of change and fault prone classes. In LR, there are two stepwise selection methods to obtain an optimal subset of independent variables, forward selection and backward elimination [76]. Forward selection examines the variables that are selected one at a time for entry at each step. The backward elimination method includes all the independent variables in the model and the variables are deleted one at a time from the model until a stopping criteria is fulfilled. We have used the forward stepwise selection method in this work.

The simple logistic model is based on a linear relationship between the natural logarithm (ln) of the odds of an event and a numerical independent variable. We derive the univariate LR formula as follows [76]:

$odds = \frac{P}{1-P}$ ...(1)

where, P = probability of the occurrence of an event

Therefore, in this work, P is the probability of a class being change or fault prone.

According to the definition of a simple logistic model:

ln (odds) = $\alpha + \beta x$ …(2)

where, x is the independent variable, i.e. an OO metric; α and β are the Y-intercept and the slope, respectively.

From (1) and (2), we get

$$P = \frac{e^{g(x)}}{1+e^{g(x)}}$$

Where, g(x) = $\alpha + \beta x$; α is also known as constant and β as estimated coefficient

This univariate formula can be extended to general multivariate formula as:

$$P = \frac{e^{\alpha+\beta_1 x_1+\beta_2 x_2+\ldots+\beta_n x_n}}{1+e^{\alpha+\beta_1 x_1+\beta_2 x_2+\ldots+\beta_n x_n}}$$

Where, x_i, ($1 \le i \le n$) are the independent variables (OO metrics)

The following statistics are reported for each significant metric:

- Maximum likelihood estimation (MLE) and coefficients (β_i): MLE is a statistical method for estimating the coefficients of a model. The likelihood function measures the probability of observing the set of dependent variable values. MLE involves finding the coefficients that make the log of the likelihood function as large as possible. The larger the value of the coefficients, the larger the impact of the independent variables on the dependent variable [5, 76].

- Odds ratio: The odds ratio is calculated using β_i and is given by the formula, R= e^{β_i} where β is coefficient. It is defined as the probability of the event divided by the probability of the nonevent [5]. In this study, the event corresponds to probability of a class being change prone or fault prone, whereas the nonevent is the probability of a class not being change prone or fault prone. An ‘odds ratio’ with a value of two means that the dependent variable is multiplied by two when the independent variable increases by one unit [5, 76].

- Statistical significance: Statistical significance measures the significance level of the coefficient. The larger the statistical significance, the lower the estimated impact of the independent variables on the dependent variable. The parameter used to measure the statistical significance is two-tailed p-value whose value is determined using the Wald test (W). The value of Wald test for an independent variable is the ratio of its coefficient to its standard error. For example, let the coefficient of an independent variable be 0.111 and its standard error be 0.024.Then, W = 0.111 / 0.024 = 4.61, and the two tailed p-value (parameter to measure significance) is P(|z| > 4.61), where z denotes a random variable following the standard normal distribution. (For details, refer [76]).

Multicollinearity is a statistical phenomenon in which the independent variables are highly correlated with each other and hence, contribute as disturbance in the datasets. It is an undesirable situation as it makes some of the independent variables statistically insignificant while they should be otherwise significant. Thus, we performed a test of multicollinearity to determine if the independent variables are correlated amongst each other. Principal Component (PC) method is applied on the independent variables to find the maximum eigenvalue (e_max) and minimum eigenvalue (e_min). Then we calculate the conditional number given as $= \sqrt{((e\_max)/(e\_min))}$. If the value of the conditional number is less than 30, we say that there does not exist multicollinearity between the variables [11].

2.6.2 Decision Tree

DT can be used for building classification as well as regression models. The resultant models are in the form of a tree structure consisting of the internal nodes and the terminal nodes. The internal nodes also known as decision nodes represent the independent variables. The branches from the internal nodes represent the possible values that these independent variables can have. The terminal nodes represent the values of the dependent variable. The topmost decision node, known as the root node corresponds to the best attribute that discriminates the dependent variables very well. In thsi work, we have used J48 and CRT.

2.6.3 Bayesian Networks

BN are used when one wants to represent the probabilistic relationship between a set of variables [115]. Graphically, it is represented as directed acyclic graph whose nodes correspond to the variables and edges show the influence of one variable on the other. A directed edge from variable V_i to V_j shows that V_i is parent of V_j. Bayesian learners build the models using Bayes rule given as follows:
$Prob(B/A) = \frac{Prob(A/B) * Prob(B)}{Prob(A)}$

- Where, P(B|A) is the posterior probability of dependent variable given the independent variable.
- P(B) is the prior probability of dependent variable.
- P(A|B) is the probability of independent variable given the dependent variable.
- P(A) is the prior probability of independent variable.

Thus, bayesian learners are used for classification by finding the posterior probability of the dependent variable given the values of the other variables. The simplest bayesian learner is NB which we have used in this work. The detailed explanation of NB is given in chapter 7.

2.6.4 Ensemble Learners

Ensemble learning is the process of combining multiple classifiers to solve a computational problem. Among number of classifiers available, it is difficult to make a choice of the appropriate classifier for our problem. Combining multiple classifiers helps to reduce the chances of making a poor or wrong selection. It may or may not improve the performance over a single classifier, but it certainly reduces the risk of poor selection. Besides this, ensemble learning can be used in both the cases; when we have large volume of data or when the data is too little. When the data available is too large for a single classifier to be trained, we can partition the data into subsets and allow different classifiers to be trained on each subset. Then, we can combine the results using some specified rules. On the other hand, when the data is too small, we can use bootstrapping mechanism, wherein we draw the data with replacement and apply a classifier to each sample. There are various EL that are used in this work such as RF, bagging, LB and AB.

2.6.5 Artificial Neural Networks

ANN comprises a network of simple interconnected units called neurons or processing units. ANN consists of three layers: the input layer, hidden layer and output layer

[71]. The first layer has input neurons which send data via connections called weights to the second layer of neurons and then again via more weights to the third layer of output neurons. Thus, the input neurons are connected to every neuron of the hidden layer but are not directly connected to the output neurons. Error back - propagation algorithm is used for training ANN. Error back-propagation learning consists of two passes: a forward pass and a backward pass. In the forward pass, an input is presented to the neural network and its effect is propagated through the network layer by layer. During the forward pass the weights of the network are all fixed. An error is composed from the difference between the desired response and the system output. During the backward pass, the weights are all updated and adjusted according to the error computed. This error information is fed back to the system which adjusts the system parameters in a systematic fashion (the learning rule). The process is repeated until the performance is acceptable [71]. MLP is an example of an ANN which we have used in this work.

2.6.6 Support Vector Machines

SVM is a learning technique that is used for classifying unseen data correctly. For doing this, SVM builds an optimal hyperplane which separates the data in such a way that the cases with one category are on one side of the hyperplane, whereas the cases with the other category are on the other side of the plane into different categories [132].

The data may or may not be linearly separable. By "linearly separable" we mean that the cases can be completely separated (i.e., the cases with one category are on

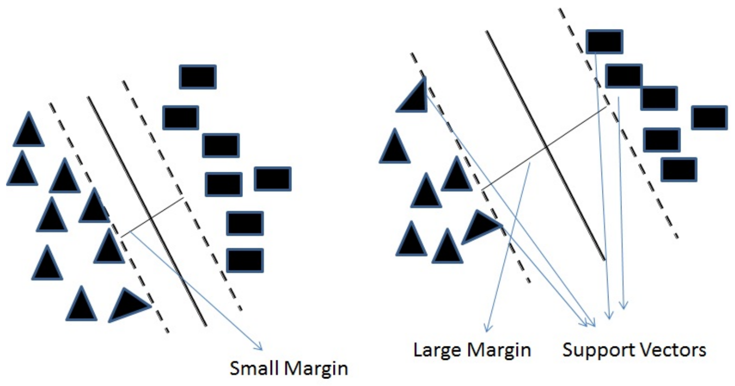


Figure 2.2: Support Vector Machine

the one side of the hyperplane and the cases with the other category are on the other side). For example, figure 2.2 shows a data where examples belong to two different categories - triangles and squares. Since these points are represented on a 2- dimensional plane, a 1-dimensional line can separate them. To separate these points into 2 different categories, there are an infinite number of lines possible. Two possible candidate lines are shown in figure 2.2. However, only one of the lines gives a maximum separation/ margin and that line is selected. “Margin” is defined as the distance between the dashed lines (as shown in figure 2.2), which is drawn parallel to the separating lines. These dashed lines give the distance between the separating line and closest vectors to the line. These vectors are called support vectors. SVM can also be extended to the non-linear boundaries by using the kernel trick. The kernel function transforms the data into a higher dimensional space to make the separation easy [44]. The commonly used kernel function are linear, polynomial, Radial Basis Function (RBF) and sigmoid. Since, RBF is the most recommended kernel function [132], we have used RBF in this study to predict change prone classes. RBF handles non linear relationships between the dependent an the independent variable by mapping non linear data into a higher dimension space.

2.7 Empirical Data Collection

The data collected to be used for empirical validation is either from university systems, commercial systems or from OSS. OSS are developed with a different approach and methodology where users are treated as co-developers. Thus, it becomes easy to customize or upgrade OSS. Besides this, they are available free of cost and there are no copyright issues. Due to these numerous advantages, we have explored the use of various OSS in this study.

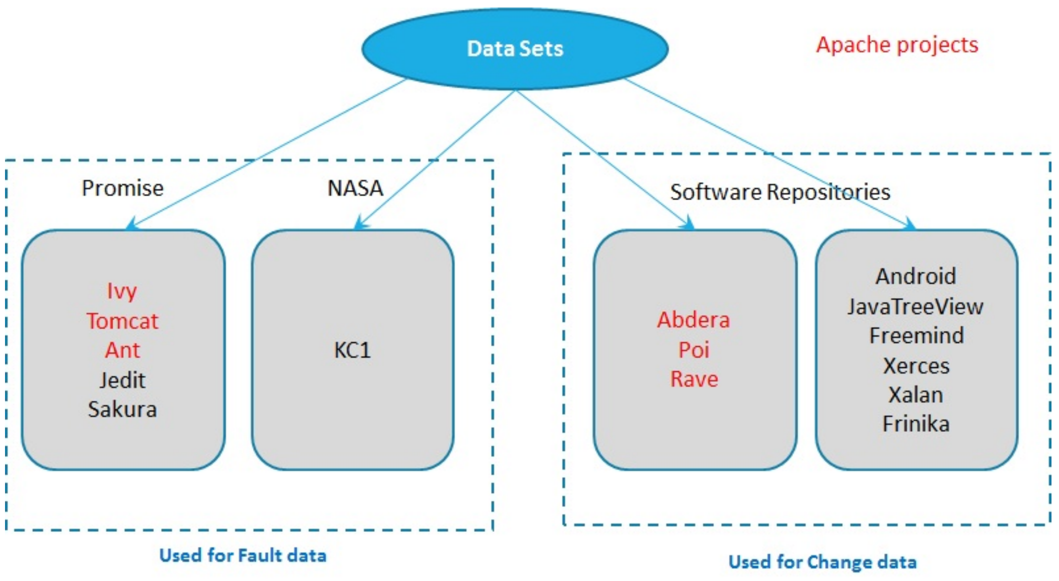


Figure 2.3: Software Used for Empirical Data Collection

Figure 2.3 shows different software we have used. All the fault data used in this study has been obtained from PROMISE repository. Whereas, for the change data, we have used two popular OSS repositories (CVS and GIT) for obtaining the datasets.

CVS (Concurrent Versioning System) and Git are version control systems used for storing code and associated data. In this section, we explain in detail the steps followed to collect the change data for CVS and Git based OSS. We also list various software used in this study along with their important characteristics.

2.7.1 Change Collection Process

We followed the steps given below in order to collect OO metrics and changes between different releases of the software [103, 153]. The diagrammatic representation of the process followed for change collection has been depicted in figure 2.4.

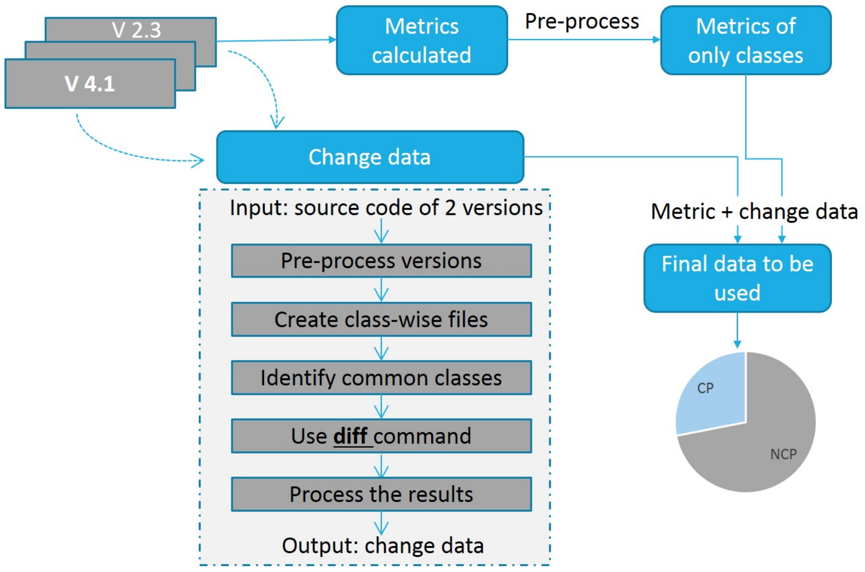


Figure 2.4: Change Collection Process

1. Compute Metrics: With the help of the tool, 'Understand for Java' (`http://www.scitools.com/`), we collected various OO metrics for the first release of the software, among the two releases under consideration. The tool gives

metrics for all the Java files, constructors, packages, classes and various other constructs. As we are concerned only with classes, we keep only the metrics for all the classes and discard all the other metrics of other constructs such as files. Besides this, the tool also generates metrics for various anonymous or unknown classes that cannot be accessed. Such classes are also removed from the analysis.

2. Pre-process releases: After downloading the source code of both the releases, we extracted the full names (i.e. package name +class name) of all the classes of both the releases. We observed that some classes in different Java files have same names, although their implementations may be different. For such classes, the computations of metrics (step 1) are biased and not correct. Hence, such classes are removed and not considered.

3. Extraction of Common Classes: The main aim behind this work is the prediction of changes in the classes of later release based on the changes in the classes of previous release. In other words, it is clear that we will be able to predict changes only in those classes of later release that also appeared in previous release. Therefore, we considered only the common classes between the two releases under consideration. This is done by comparing the full names (i.e. package name+ class name) of all the classes of the previous release with the classes of the later release.

4. Comparing classes with the help of a Tool: We compared the source code of each class of the previous release with its corresponding class (class with the same name) in the later release and collected change as the number of source

lines of code (SLOC) added, deleted and modified in each Java class of the previous release with respect to the class of same name in the later release. For doing this, a tool (Configuration Management System (CMS)) [96] with Graphical User Interface (GUI) is used. This tool has been developed in Java programming language and executes efficiently on all releases of Windows operating system.

5. Computation of Change Statistics: Based on the number of source lines of code added, deleted and modified, 'TOTAL CHANGE' for a class is calculated as follows:

    - Each added or deleted line is counted as one SLOC change.
    - Each modified line is counted as two SLOC change i.e. one deletion followed by one addition.

      Then we defined a binary variable 'CHANGE', which is assigned a value 1 if 'TOTAL CHANGE' >1 or 0 otherwise.

6. Collection of Data Points: A data point or an instance refers to a class along with all its metrics values and the value of binary variable 'CHANGE'. Thus, metric values for the classes common between the two releases obtained from step 1 are combined with the values of binary variable 'CHANGE' obtained from step 5 to generate data points.

2.7.2 Open Source Software Used

The use of a large number of popular and widely used OSS has become possible because of the establishment and the wide prevalence of OSS development. The software used in the study for change and fault data along with some of their important characteristics are listed in tables 2.2 and 2.3 respectively. For each of the software, we have stated the total number of classes and the number of classes which are found to be changed or are faulty in the later release with respect to the previous release.

Table 2.2: Software Used for Change Data

Software	Release	Total classes	Classes changed	Type/Function
Android	2.3	5018	1973	Mobile Operating System
	4	5710	1816	
	4.1	8518	1609	
	4.2	8455	1557	
	4.3	9218	1554	
Freemind	0.9.0	656	29	Mind mapper & hierarchical editor
	0.9.1	608	118	
	1.0.0	668	555	
Xerces	2.9.0	568	120	Create & maintain XML parsers
	2.9.1	572	294	
	2.10.0	614	74	
Frinika	0.2.0	248	126	A complete music workstation
Xalan	2.6.0	84	554	EXtensible Stylesheet Language Transformations (XSLT) processor
Abdera	1	679	624	Implementation of the Atom Syndication and Atom Publishing Protocol. They are the standards for creating, editing and publishing various web sources
	1.1	677	8	
	1.1.1	686	634	
	1.1.2	685	635	

Software	Release	Total classes	Classes changed	Type/Function
POI	3	1515	1276	Used for creating and manipulating various file formats
	3.6	2088	1988	
	3.7	2472	2212	
	3.9	2786	2706	
Rave	0.19	607	32	Provides an extensible platform for using, integrating and hosting various OpenSocial and W3C Widget related features, technologies and services
	0.20.1	642	628	
	0.21.1	676	648	
	0.22	685	225	
JavaTreeView	1.0.3	97	40	Cross-Platform Gene Expression Visualization Tool

Table 2.3: Software Used for Fault Data

Software	Release	Total classes	Faulty classes	Type/Function
Ivy	2	352	40	Powerful dependency manager that manages dependencies of any kind
Ant	1.7	745	166	Java library and command-line tool for automating software processes
Tomcat	6	858	77	Provides pure Java HTTP web server environment for Java code to run in
JEdit	4.3	492	10	Text editor for programmers
Sakura	2.0.2.0	80	47	Japanese text editor for programmers
KC1	-	145	58	Proprietary software of NASA (NASA Metrics Data Program)

2.8 Data Analysis and Pre - processing

After the data is collected, the next step is to analyze and pre - process the data. For analyzing the data, we have calculated various descriptive statistics and for pre -

processing the data, feature reduction techniques are used.

2.8.1 Descriptive Statistics

Various descriptive statistics are used for describing and analyzing the research data. Descriptive statistics concern development of certain measures to summarize data. In other words, they allow to have a thorough knowledge of the data and draw important conclusions from it. The important statistics measures used in this study for describing the central tendency of the values are mean and median. Median is the numeric value that separates the higher half of the data from the lower half, whereas mean represents the average value. It is a better representation of the data when the data contains the outliers. For example, let weights of 10 students in a class be between 51 and 61 except for the one student whose weight is 210. In this case, the mean will be 72 and the median will be 58. Hence, the median better reflects the weight of the students than the mean. To describe the dispersion in the data, different quartiles (25% and 75%) and standard deviation are calculated. Skewness and kurtosis are used to measure the shape of the distribution. Skewness measures the degree of asymmetry, whereas kurtosis measures peakedness or flatness of the distribution. The distribution of a variable can be considered normal if its skewness and kurtosis statistics fall between -1 and +1. Besides these statistics, the most common statistics such as minimum and maximum are also calculated.

2.8.2 Data Reduction

The presence of a large number of features in an empirical data may reduce the prediction efficiency of various ML and statistical models. Thus, it is very important to reduce the dimensionality of the data which will reduce the size of the hypothesis space and will allow the models to operate more efficiently. Besides improved computational efficiency, data reduction also leads to lower cost, increased problem understanding, and improved accuracy. In this study, we have obtained a subset of features (independent variables) by eliminating the attributes that have little or no predictive information. In other words, we have used only those features which are significant in predicting the dependent variable. For this, two popularly used techniques are used, i.e. Correlation - based Feature Selection (CFS) and univariate analysis. Each of these techniques is explained briefly in this section.

Univariate Analysis

The univariate analysis is done to find the individual effect of each independent variable on the dependent variable. It eliminates the variables which are not significantly related to the dependent variables. The method to be used for univariate analysis depends on the type of dependent variables being used. The dependent variables used in this study are change proneness and fault proneness. We have used regression analysis to preselect the features which are significant predictors of the dependent variable. We say that the independent variables which are significant at 0.05 significance level can be considered as significant in predicting the dependent variable and can be used for subsequent model prediction.

Correlation - based Feature Selection

CFS is used to select the best attributes from the set of independent variables. The best combinations of independent variables are searched through all the possible combinations of independent variables. CFS evaluates the best subset of variables by considering the individual predictive ability of each feature along with the degree of redundancy between them. In other words, it helps to find a subset of independent variables that are highly correlated with the dependent variable but are not related with each other [111]. This leads to a selection of good feature set in which the number of independent variables is reduced (termed as 'data dimensionality reduction'). M.A. Hall proved that "classification accuracy using reduced feature set is equal or better than accuracy using complete feature set" [64].

2.9 Model Prediction

The basic premise of software quality prediction is that a class currently under development is change (or fault) prone if a class possessing the similar product or process metrics in some older project developed in the same environment (i.e. of similar nature) was change (or fault) prone. Thus, to predict change (or fault) prone classes in a project, the data (dependent and independent variables) of the previously developed release of the same project or the data of some other project of similar nature are used. There are various advantages of identifying change (or fault) prone classes:

- During project progress, managers can pay focused attention and resources on the classes found at risk. This will allow the managers to more evenly distribute the workload among the developers and testers.

- Designers can redesign these classes, if required. For example, if designers find that the modification in any class affects other classes to a large extent, this concludes that the coupling between the classes is high and thus, should be reduced.
- Knowing the classes at risk, the testers can prioritize their testing resources and activities (walkthrough, inspection) on such classes.

All these factors lead to substantial saving of resources and reduction of costs associated with the development and maintenance phase. We consider two mechanisms to predict change (or fault) prone classes using OO metrics:

1. Construction of models using data analysis techniques: Developing various metric models with the help of different ML and statistical methods.
2. Threshold computation: Identifying the risk indicators of various OO metrics.

We will discuss each of these methods in detail in this section.

2.9.1 Model Construction using Data Analysis Techniques

We have used supervised learning to construct various classification models. In other words, the outcome or target or dependent variable is known for training the model. Thus, the training dataset has the values of both the independent and dependent variables with the help of which the classifier learns or trains the model. After the model is trained or constructed, it is tested using the validation set which can be a subpart of the same release or some other similar dataset. Finally, the constructed model can be used to make predictions of the dependent variable for a new dataset. In other words, the constructed model can be used to predict change (or fault) prone classes of the future release of the same software or similar other software. This process of model construction is shown in figure 2.5.

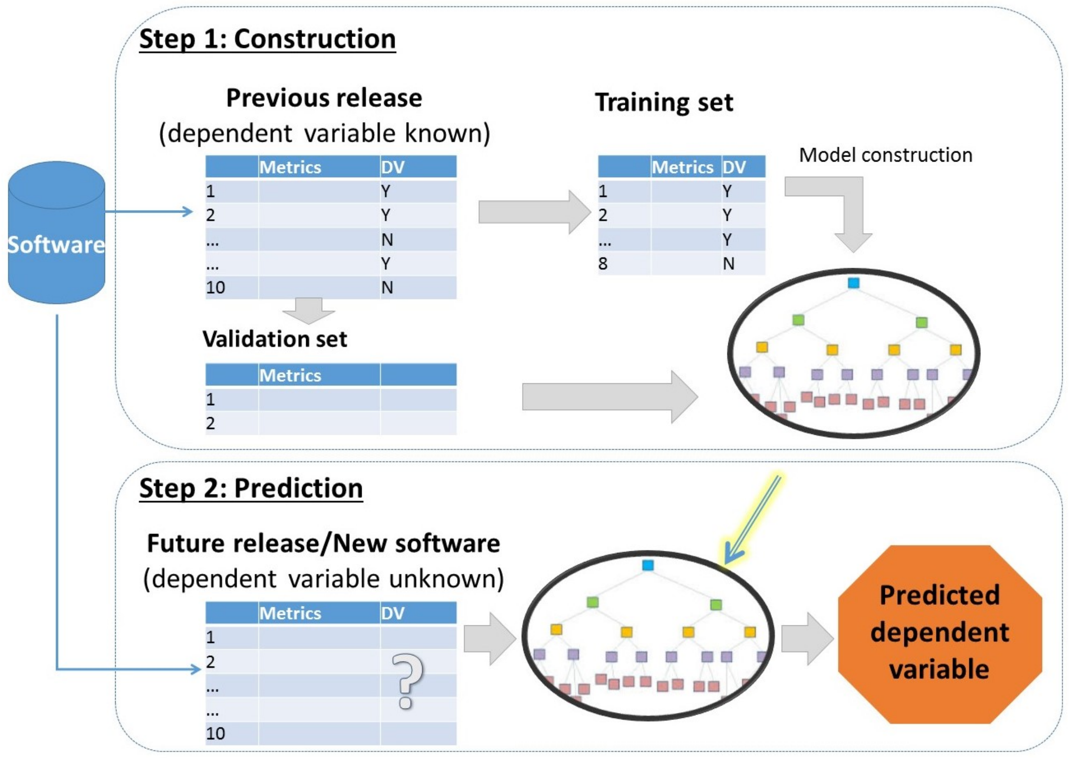


Figure 2.5: Model Construction Using Data Analysis Techniques

2.9.2 Threshold Computation

Another method to identify change (or fault) prone classes is by using the thresholds of metrics. Metrics measure internal characteristics or attributes of software such as size, coupling, cohesion etc. Once metric values are determined, there should be a technique or methodology to assess if the metric values are good or bad. For this purpose, we have identified threshold values for metrics. Bender [12] defined thresholds as “Breakpoints that are used to identify the acceptable risk in classes”. They provide an upper bound on the metric values such that the classes having metric values above the threshold values are considered to be problematic, whereas the values lower are considered to be acceptable. A class is said to be problematic, when at least one threshold for a metric is violated. Thresholds can be calculated at the initial phase of SDLC. Following are the advantages of using thresholds for identification of risky classes:

- Thresholds identify certain alarming values of the metrics which are used to find classes that fall outside the acceptable risk level.
- Management personnel’s can use them as quality benchmarks to gain insight about the quality of the software, allowing them to assess, control and compare the quality.
- Threshold values allow the designers to have a quick overlook on the metric values and alter them accordingly. In other words, the thresholds identify design anomalies and source code flaws.
- Threshold of each metric serve a different purpose. For example, if the value of coupling metric of a class is more than its threshold, this implies that the class is excessively coupled to other classes. Thus, potential candidates for redesigning the classes can be selected based on threshold values.

We have identified the thresholds for various OO metrics using two approaches: (1) Statistical approach based on LR and (2) Using the ROC curve. These are explained in detail in chapters 6 and 8 respectively.

2.10 Validation Methods

Once the model is trained, it is very essential to validate the model which allows to assess the performance of the predicted model. The trained or the predicted model can be validated on either the same software from which it is derived or a different, similar - natured software. When a same software is used for both training as well as testing the model, the procedure (method) may be referred to as internal validation. Whereas, when two different software or releases are used for training and testing the model, procedure (method) may be referred to as external validation. In this section, we have discussed both the types of validation.

2.10.1 Internal Validation

In internal validation, a single release or two different releases of the same software for both training and testing the model. When the same release is used for both training and testing, then we may obtain highly optimistic results. Thus, the entire data set (release) is not used when training a model. A subsample of data is used for training the model and this subsample is called as the training data. The data that was removed is used to validate the model (by acting as “blind” data) and is known as the validation data, or test data. The data set can be split into the training and validation data in the following three ways:

1. Holdout cross-validation: Holdout cross - validation is the simplest kind of cross-validation. Dataset is divided into two subparts such that some of the observations form the validation data and the remaining observations are retained as the training data. Generally, one third of the dataset is used as the validation data. This method is used when the dataset is large, i.e. the number of samples is large [71]. However, its evaluation can have a high variance since it heavily depends on which data points form the training data and which form the validation data. Thus, the evaluation is significantly different depending on how the division is made.

2. K-cross validation: In K-cross validation, the dataset is partitioned into K subsamples. Of the K subsamples, one subsample is used as the validation data for testing the model and the remaining K – 1 subsamples are used for training the data. This process is repeated K times such that each of the K subsamples is used exactly once as the validation data. The K results from the folds then can be combined to produce a single estimation. The advantage of K-cross validation is that the evaluation does not depend on how the data is partitioned. Every observation is in the testing set exactly once and is in the training set K-1 times. The variance of the resulting estimate is reduced as K is increased. The disadvantage of this method is that the training algorithm has to rerun from scratch K times, which means it takes K times as much computation to make an evaluation.

3. Leave-one-out cross validation: Leave-one-out cross validation method uses a single observation from the dataset as the validation data and the remaining

observations as the training data. This process is repeated such that each observation in the dataset is used once as the validation data. This is the same as K-fold cross-validation where K is equal to one. When the number of observations is very limited, one may use this form of cross validation [71].

In this work, the size of the dataset is greater than 80 data points, hence leave-one-out method has not been used.

2.10.2 External Validation

In external validation, we may use different releases of the same software of two different, similar natured software for training and testing the model. Thus, there are two types of external validation techniques: inter - project and inter - release validation.

Inter - Release Validation

In inter - release validation, we use two different releases of the same software for both training and testing the model. For example, a model trained using a release ’r’ of a software is validated on the upcoming releases ’r+1’, ’r+2’ etc. This concept is depicted in figure 2.6.

Inter - Project Validation

In inter- project validation, the models are trained using the historical data of some other software. For example, to predict changes in a software B, we train the models using the data of software A, and then test them on the software B. We can observe

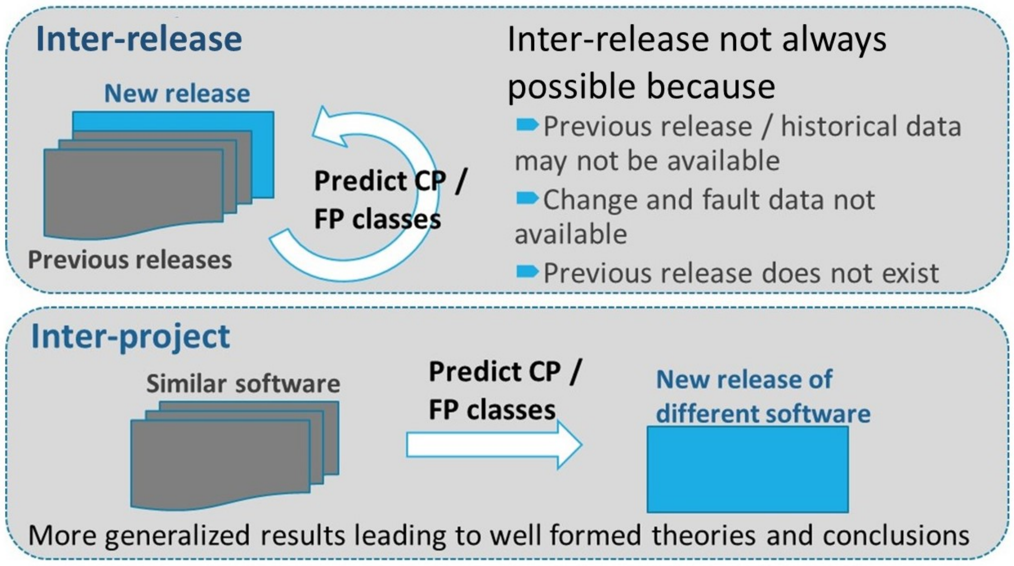

Inter-release
New release
Previous releases
Predict CP / FP classes
Inter-release not always possible because
Previous release / historical data may not be available
Change and fault data not available
Previous release does not exist
Inter-project
Similar software
Predict CP / FP classes
New release of different software
More generalized results leading to well formed theories and conclusions

Figure 2.6: Validation Techniques

that training and testing are performed on the different software, instead of using the same software. This method of validation leads to more generalized results. This concept is diagrammatically explained in figure 2.6.

2.11 Performance Measures Used

The performance of binary prediction models is typically evaluated using confusion matrix shown in table 2.4.

Table 2.4: Confusion Matrix

Observed	**Predicted**	
	Positive	Negative
Positive	TP	FN
Negative	FP	TN

The terms in the table 2.4 are defined as follows:

- TP (True Positive): It is the count of the correctly predicted positive cases
- FP (False Positive): It is the count of the incorrectly predicted positive cases
- TN (True Negative): It is the count of the correctly predicted negative cases
- FN (False Negative): It is the count of the incorrectly predicted negative cases

In this study, a positive case refer to a class being change prone and a negative case refer to a class being non - change prone. We define various performance measures used for assessing binary classifiers with the help of TP, FP, FN and TN.

1. Sensitivity: Sensitivity is also known as recall, True Positive Rate (TPR) and Probability of Detection (pd). It is defined as the ratio of the number of classes correctly predicted to be change prone to the number of actual change prone classes.
   Mathematically, Sensitivity = TP/(TP + FN)
2. Specificity: Specificity is defined as the ratio of the number of classes correctly predicted to be non - change prone to the number of actual non - change prone classes.
   Mathematically, Specificity = TN/ (TN + FP)
   It can also be defined in terms of False Negative Ratio (FNR) as:
   Specificity = 1-FNR,
   Where FNR (also known as probability of false alarm, pf) is defined as the ratio of the number of classes incorrectly predicted to be change prone to the number of actual non- change prone classes. Mathematically, FNR= FP/ (FP+TN)

3. Precision: Precision is defined as the ratio of the number of classes correctly predicted to be change prone to the number of classes predicted to be change-prone.

   Mathematically, Precision = TP/ (TP+FP)

4. Accuracy: Accuracy is also known as correctness. It is defined as the ratio of the number of classes correctly predicted to the total number of classes.

   Mathematically, Accuracy = (TP+TN)/ (TP+FN+FP+TN)

5. Balance: Balance combines the probability of detection (pd) and the probability of false alarm (pf). The point (pf=0 and pd=1) is the most suitable point on the ROC curve. Thus, balance is defined as the Euclidean distance from the point <pf=0 and pd=1> to a pair of <pf,pd>. For the purpose of convenience following 2 steps are taken [110]:

   (1) We know that the maximum possible distance across the ROC square is √2. Thus, balance is normalized by this maximum distance.

   (2) After normalization, the terms obtained in step (1) is subtracted from 1.

   Mathematically, Balance = $1 - \frac{\sqrt{(0-pf)^2+(1-pd)^2}}{\sqrt{2}}$

6. G-mean: G-mean is the geometric mean of the accuracy of positives (a+) and the accuracy of negatives (a-). This evaluation technique was suggested by Kubat and Matwin [83].

   Mathematically, G-mean = $\sqrt{(a+) * (a-)}$

   where, a+ (accuracy of positives): denotes the ratio of number of classes correctly predicted to be change prone to the number of classes predicted to be change prone. Mathematically, a+ = TP/(TP+FP) a- (accuracy of negatives): denotes the ratio of number of classes correctly predicted to be non - change prone to the number of classes predicted to be non - change prone. Mathematically, a- = TN/(TN+FN)

7. Receiver Operating Characteristics Curves (ROC): The ROC curve, which is defined as a plot of sensitivity on the y-coordinate versus its 1-specificity on the x- coordinate is a commonly used way to visualize the performance of a binary classifier [44]. According to the definition of sensitivity and specificity, we understand that high values of sensitivity and specificity are desirable. At the same time, we also need to ensure sensitivity is approximately equal to specificity. Thus, the ROC curve helps to find an optimal cut-off point that maximizes both sensitivity and specificity [44]. This allows to maintain a balance between the number of classes correctly predicted to be change prone and the number of classes correctly predicted to be non - change prone. The Area Under the curve (AUC) derived from ROC analysis is used to measure the performance of a model. The area measures discrimination, i.e. the ability of the models to correctly classify a class as change prone or non - change prone. AUC lies between 0 and 1 and higher the value of AUC, better is the predictive capability of the model [52].

An ideal model should have high values for all performance measures. Their values are distributed between 0 and 1, with 1 being a perfect score and 0 being the worst.

2.12 Significance Tests

In this work, we use statistical tests to find the statistical significance of the ML techniques.

2.12.1 Friedman Test

Friedman test is a nonparametric test that is used to rank a set of k treatments over multiple data instances or subjects [57]. The Friedman test is based on the following hypothesis:
Null Hypothesis (Ho): There is no statistical difference between the performance of the compared techniques.
Alternative Hypothesis: (H1): There exists statistical difference between the performance of the compared techniques.
Following steps are used to compute the Friedman test statistic (χ2):

1. Determine ranks: Organize the data values of all the treatments for a specific dataset in descending (high to low) order. Allocate ranks to all the values. Rank 1 is assigned to the best performing treatment. In case of two or more observations of equal values, assign the average of the ranks that would have been assigned to the observations.

2. Compute total ranks: Compute the total of ranks allocated to a specific treatment on all the datasets. The rank total for k treatments is denoted by R_1, R_2, … R_k.

3. Compute χ2-statistic : The χ2-statistic is computed by the following formula:

$\chi 2 = \frac{12}{nk(k+1)} \sum R^2 - 3n(k+1)$

Where, R = sum of ranks (Sum) for every technique, n = number of data sets, k = number of compared techniques

If the value of χ2 is in the critical region with specific level of significance, then the null hypothesis is rejected and it is concluded that there is difference between the performance of the two techniques, otherwise null hypothesis is accepted.

2.12.2 Post - hoc Analysis

There are two statistical tests used in this test for conducting post - hoc analysis. These tests are explained in this section.

Wilcoxon Signed - Rank Test

Wilcoxon signed - rank test is a non - parametric test that performs pairwise comparisons of the difference in performance of the techniques [148]. It may or may not be used as a post - hoc test. The Wilcoxon signed - ranks test is based on the following hypothesis:

Null Hypothesis (Ho): There is no statistical difference between the performance of the two techniques.

Alternative Hypothesis (H1): There exists statistical difference between the performance of the two techniques.

It is defined as [41]:

$R^+ = \sum_{m_i>0} rank(m_i)$

$R^- = \sum_{m_i<0} rank(m_i)$

Where, m_i is the difference between performance measure of first technique from the second technique when applied on n data sets. The differences are ranked based on their absolute values. Here, R^+ is the sum of ranks for the given data sets where second technique outperforms the first technique and R^- is the sum of ranks for the given datasets where first ML technique outperforms the second technique.

Following are the steps used for calculating the rank:

1. Exclude the pairs where the absolute difference is 0. Let n_r be the reduced number of pairs.

2. Assign rank to the remaining n_r pairs based on the absolute difference. The smallest absolute difference is assigned a rank 1.

3. If there is a tie in the absolute difference, then ties receive a rank equal to the average of the ranks they span.

Finally value of z is calculated as:

$$Z= \frac{Q-\frac{1}{4}n_r(n_r+1)}{\sqrt{\frac{1}{24}n_r(n_r+1)(2n_r+1)}}$$

where, Q = min(R^+, R^-)

Similar to the Friedman test, if the Wilcoxon value (Z) is in the critical region with specific level of significance, then the null hypothesis is rejected and it is concluded that there is difference between the performance of the two techniques, otherwise null hypothesis is accepted.

Nemenyi Test

Nemenyi test is used as a post - hoc to examine the statistical difference between the pairs of different techniques. It can be used after the application of Friedman test, if the null hypothesis of the corresponding test is rejected. It compares all the techniques with each other and investigate whether the performance of two techniques differ significantly [41]. It is based on the Critical Distance (CD) which is computed as:

CD= $q_a\sqrt{\frac{k(k+1)}{6n}}$

Where, n=number of datasets, k=number of algorithms, q_a= critical values (studentized range statistic divided by $\sqrt{2}$)

The computed CD value is compared with the difference between average ranks allocated to two techniques. If the difference is equal to or greater than the CD value, the two techniques differ significantly at the chosen significance level (i.e. 0.05).

Chapter 3

Predicting Change Using Software Metrics: A Review

3.1 Introduction

Maintaining the quality of the software is very important. We know that the changes in software are inevitable due to diverse reasons such as change in user requirements, competitive pressure, increasing customer demands etc. Due to the changes, maintenance cost keeps on increasing. With the help of the OO metrics and the change data collected from a similar project or previous release of the same project, prediction models can be developed. Early prediction of change prone classes allows the developers to pay focused attention on such classes by judiciously allocating the resources, allows the designer to re-design the classes and allows the tester to focus his testing resources and testing activities (such as inspection, walk through) on such classes. Therefore, software change prediction leads to improved quality.

It is necessary to systematically summarize and analyze the empirical evidence obtained on change prediction studies from the existing literature. Thus, the primary goal and contribution of this chapter is to support the research on change prediction through an extensive review of the relevant studies. In other words, this review synthesizes existing work in the area of software change prediction which will allow researchers and practitioners to have a fair evaluation of all the studies. They will be able to examine the previous studies from different viewpoints: metrics, data analysis techniques, datasets, and experimental results perspectives. We obtained significant conclusions than is possible from individual studies, or as a prelude to further research activities [80]. The RQs formulated in the review allowed us to identify gaps in the current technology. We have provided future guidelines to software practitioners and researchers to overcome the gaps. Following this section, we have explained in detail the methodology used in conducting this review.
The results of this chapter are published in [97, 101].

3.2 Research Methodology

A systematic review consists of a number of discrete steps. We carried the review by following the procedure as given by Kitchenham [80]. The various steps taken to conduct this review are listed below and are also represented diagrammatically in figure 3.1.

Each of the following steps are explained in detail in the subsequent sections.

- Step 1: Identifying the need of the review

- Step 2: Identifying the review process
- Step 3: Reporting and documenting the review
- Step 4: Reporting review results
- Step 5: Concluding the review

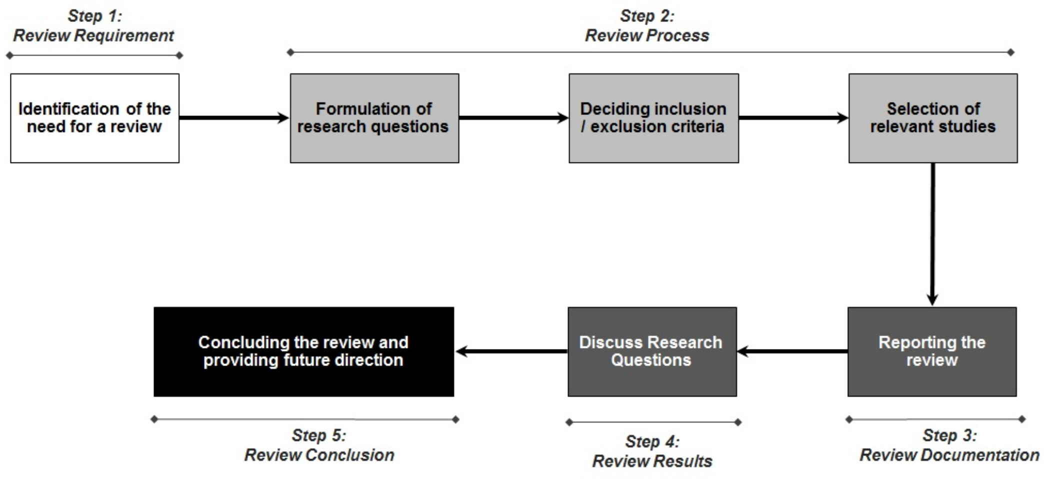

Step 1:
Review Requirement
Step 2:
Review Process
Identification of the need for a review
Formulation of research questions
Deciding inclusion / exclusion criteria
Selection of relevant studies
Concluding the review and providing future direction
Discuss Research Questions
Reporting the review
Step 5:
Review Conclusion
Step 4:
Review Results
Step 3:
Review Documentation

Figure 3.1: Framework of the Review

The step to identify the correct process to carry out the review is explained in section 3.3. The next step involves reporting or documenting the review as explained in section 3.4. Following this, we have represented the results of the review in section 3.5. Finally, the review is concluded and future directions are provided in section 3.6.

3.3 Review Process

It is important to follow an appropriate and correct procedure to carry out a review. Depending on the type and nature of research, the review process may differ. For this research, the review process which has been followed consists of three steps: (1) Formulating RQs, (2) Deciding inclusion/exclusion criteria and (3) Selecting relevant studies. Each of these steps is discussed in this section.

3.3.1 Formulation of Research Questions

Formulation of RQs is very important to carry out a research. It is critical to have clarity about questions which need to be answered so that a focused effort can be undertaken. The RQs allow the researchers, practitioners and the readers to know what the review is intended to answer. These questions are primarily of interest to researchers. In this review study, we have addressed the following issues:

- RQ1: What types of metrics are most commonly used in the change prediction?
- RQ2: What types of datasets are most widely used for change prediction?
- RQ3: What type of machine learning techniques are used for change prediction?
- RQ4: What are the significant predictors of change proneness?
- RQ5: Have appropriate performance measures been used to evaluate the performance of the models?
- RQ6: Have appropriate statistical tests been used to measure the performance?
- RQ7: What are the risk indicators for various OO metrics?
- RQ8: What validation techniques are used to validate the models?

3.3.2 Inclusion and Exclusion criteria

Deciding the appropriate inclusion and exclusion criteria is very important to assess each potential study. The selection or rejection of a study in the review depends on the inclusion and exclusion criteria respectively. The formulation of the inclusion and exclusion criteria should be based on the RQs. In this review, following inclusion and exclusion criteria is used:

Inclusion Criteria

- Empirical studies establishing relationship between change proneness and metrics.
- Empirical studies where change prediction is associated with change impact analysis or change propagation.
- Empirical studies using any type of software metrics (product metrics, process metrics etc.) to predict change proneness.
- Empirical studies using ML statistical techniques for software change prediction.

Exclusion Criteria

- Empirical studies based on dependent variable other than change proneness or change prediction.
- Empirical studies which have considered 'change' as continuous dependent variable.

- Empirical studies where independent variables are not metrics (for example, studies where independent variables are design patterns, code smells etc. are not considered).
- Empirical studies which have used ML or statistical techniques in context other than software change prediction.
- Review or survey studies for change prediction.

3.3.3 Selection of Relevant Studies

We included all the relevant studies that are filtered according to the defined inclusion criteria. Further, for searching all the appropriate studies, we defined a search strategy. We obtained a subset of studies that are selected using the formed search strategy. Each of the studies in this subset is thoroughly read to obtain the final set of studies to be used in this review. The detailed explanation of the steps taken to obtain the final set of studies is given below:

1. Step 1 : Search strategy to get 'Initial list' of studies

   A comprehensive search is conducted to identify all the studies whose title or abstract contains some of the relevant keywords such as change prediction, change proneness, change impact, change propagation, software change etc. After searching the studies based on these individual keywords, new keywords are also formed by combining alternative and synonyms by Boolean expression 'OR' and main words by 'AND'. For example, Change AND (Prediction OR proneness) AND (Machine learning OR statistical OR data analysis). At this step, the initial set of studies is obtained. For each of the studies in the initial set, the abstracts are scanned and read to judge the relevance of the studies. After reading the abstracts, we found that some of the studies should not be included in the review according to our inclusion criteria. For the initial search, we have visited various research related digital portals such as IEEE Explore, Springer, Science direct, Wiley and ACM digital library. We searched the studies in various journals and conference proceedings of repute [135].

2. Step 2: Selection of 'Final Set' of studies

   Once the potentially relevant primary studies have been obtained, they are assessed for their actual relevance. The final set of studies to be included in the review is obtained after the full texts have been retrieved and read. In other words, full copies of these studies are obtained and again reviewed by two senior assistant professors (having doctorate degree). The introduction and conclusion sections of the studies selected in the initial stage are read thoroughly. It is useful to maintain a list of excluded studies identifying the reason for exclusion. At the end of this step, we found 21 relevant studies related to our area of change prediction. Among these 21 relevant studies, we identified the following primary studies closest to our area of research:

   - Romano and Pinzer [124] evaluated the use of OO metrics to predict the relationship between these metrics and change prone classes.
   - He et al. [73] investigated the importance of cross project predictions for predicting fault proneness.
   - Shatnawi [129] identified threshold values of OO metrics to predict fault

prone classes of software.

3.4 Review Documentation

Documenting and reporting the review in an efficient manner is very important. There are various approaches or techniques which can be used for representing the results of the review in the most efficient manner. The most appropriate approach which can be selected for qualitative and quantitative synthesis depends on the type of RQ being addressed /citemalhotra15. Following are the tools which are used in this review for summarizing and representing the important information and results [95]:

1. Tabulation: It is the most common and popular approach for representing qualitative and quantitative data. The important details about each study in the review can be summarized in the tabular form. For example, the main concept behind each study, the outcome or the results of the study, values of various performance measures , the design of the study etc. can be presented in the tables. The information represented in the tabular form is easy to be interpreted and analyzed. In this review, we have gained important insights and represented them in tables 3.1, 3.2 and 3.3. Table 3.1 focuses on the main objective along with the results or main findings of each study in the review. In table 3.2, the studies are summarized with respect to various parameters such as the metrics used, data analysis techniques (ML or statistical) used, the performance measures used and software used. This will allow to have an overview of the metrics which are commonly used in research, the various data analysis techniques used to construct the models, the performance measures used to evaluate

the performance of the models and the software used for empirical validation. Table 3.3 lists the type of metrics and software repositories.

2. Visual diagrams: There are numerous diagrams which can be used to graphically represent important information and ideas such as bar charts, pie charts, line graphs, box plots etc. To represent some mathematical data or information, visual diagrams are preferable over the textual format as they allow to better understand, retain and analyze the results. In this review, we have used pie charts to mathematically represent the answers to some of the RQs.

3. Textual descriptions: The answers to various RQs can be addressed in the textual form as well. While giving textual descriptions, the main findings and the outcomes should be emphasized without going into much details. In this review, we have provided answers to various RQs in the textual format which are supplemented by either the visual diagrams or the tables. The textual descriptions provide a quick overview which briefly describe answers to each question.

Table 3.1: Brief Description

S.No.	Study	Objectives	Results
1.	[89]	To understand the relationship between size and change-prone classes.	Large classes are more change-prone than small classes.
2.	[90]	To analyse the stable constructs which remain unchanged, and the constructs which change often.	The class and inheritance structure, software entities' names and interfaces to methods are stable and remain unchanged.
4.	[23]	To investigate the use of coupling measurement to identify the classes likely to contain ripple changes when another class is being changed.	A coupling based model can indicate class pairs with higher ripple effect probability.

S.No.	Study	Objectives	Results
5.	[32]	To study the effect/impact on the system when a change to a system is made.	A relation between WMC, a design metric, and the mean change impact of a class is established. The higher the WMC value was, the higher was the mean change impact.
6.	[8]	To investigate whether dynamic coupling measures are significant indicators of change proneness and are complementary to existing static measures. Also to show that they capture different properties than simple size effects.	Some dynamic coupling measures are significant indicators of change proneness and they complement existing coupling measures based on static analysis.
7.	[138]	To evaluate the likelihood that each class will change in future.	There is a correlation between the probabilities extracted from the model and the actual changes in a system.The accuracy of the proposed model is better than the model which is based on past data.
8.	[1]	To analyse and predict changes impacts in OOsystems.	The four ML techniques used verified the hypothesis.
9.	[127, 128]	To assess the probability that each class will change in a future generation.	The proposed probabilistic approach is simple and accurate in the comparison with existing methods in the literature.
10.	[66]	To calculate BDM to predict change-proneness in UML 2.0 models.	BDM is an useful indicator and provides improved predictive model compared to the model considering only CK metrics [36].
11.	[153]	To investigate the confounding effect of class size on the relationship between OO and change proneness.	Confounding effect of class size exists and should be taken into account to avoid false results.

S.No.	Study	Objectives	Results
12.	[67]	To develop BDM to measure the behavioural aspects of the software.	BDM is a useful indicator and improves accuracy of change prone class prediction over that of metrics (CK [36], Lorenz and Kidd [91], MOOD [3]) when the system contains high degree of inheritance relationships and polymorphism. But when inheritance relationships and polymorphism are less in the system, BDM made no difference in the prediction of change prone classes.
13.	[50]	To determine change prone classes and the parts which should be tested first.	Proper selection of OO metrics leads to efficient estimating of change prone classes.
14.	[92]	To find significant relationship between OO metrics and change proneness. The study considered 62 metrics and 102 Java systems.	The predictive capability of size, coupling/cohesion and inheritance metrics is moderate, low and poor respectively.
15.	[124]	To examine the predictive ability of metrics (CK [36], metrics to measure the complexity and the usage of interfaces; and two metrics (IUC and a clustering metric) to measure cohesion) to classify Java interfaces into change prone and not change prone.	The IUC metric exhibits the strongest correlation with the number of source code changes and thus, improves the performance of prediction models.
16.	[58]	To explore whether a source file will be affected by a certain type of SCC	Neural Network models can predict categories of SCC types. With the help of models, a list of change-prone files is generated ordered according to their change proneness
17.	[45]	To determine if evolution-based metrics can classify change and non-change prone classes.	Evolution metrics along with the product metrics predict change prone classes more accurately.
18.	[103]	To predict change prone classes using ML techniques and compare their performance with statistical method.	ML models and statistical model gave comparable performance.
19.	[105]	To use ANFIS to calculate the change proneness and compare with other techniques like bagging, LR and DT.	ANFIS gives the best results of all the techniques investigated.

S.No.	Study	Objectives	Results
20.	[104]	To validate CK metric suite [36] for developing change prediction model using GEP.	Change prediction can be efficiently done using GEP algorithm.
21.	[107]	To explore the capabilities of Genetic Programming (symbolic regression)for for predicting defect and change proneness of classes.	Symbolic Regression is able to predict changes and defects with high precision and recall.

ANFIS: Adaptive Neuro-Fuzzy Inference System, GEP: Gene Expression Programming, IUC: Usage Cohesion, SCC: Source Code Changes, UML: Unified Modeling Language

3.5 Review Analysis and Results

In this section, we discuss the results of the review by providing answers to various RQs formulated in section 3.3.2. Each study shortlisted to be included in this review is summarized in the tabular from in table 3.1. Table 3.1 briefly explains the objectives and the main results of each study. Thus, the overview of each study can be obtained from table 3.1. To further analyze each study, we have discussed each RQ future guidelines which can be used by researchers and practitioners in their research.

3.5.1 RQ1: What types of metrics are most commonly used in the prediction of change proneness?

In the area of change prediction, we observed that only product metrics are used. Since, process metrics are not used at all, we focus on only the product metrics. Among a number of metrics in literature such as CK metric suite [36], Lorenz and Kidd [91], MOOD [3], Li and Henry [88], QMOOD [9], etc., CK metric suite is most popular. CK metrics are object oriented metrics which measure important characteristics at the class level. Various other class level metrics are also used in literature. Types of product metrics used in the studies have been listed in table 3.2. The percentage distribution of the studies by the types of product metrics used is shown in figure 3.2. As can be seen in figure 3.2, majority of the studies (>80%) have used class level metrics. Among other types of product metrics, the usage of method and file level metrics is minimal. The component level metrics are not used at all. The remaining 20% is approximately equally shared among the studies which have used method level metrics and studies which have used more than one type of metrics. Table 3.3 lists all the metrics used in the study under the column 'Metrics Used'. Some of the studies have used large number of metrics to get more generalized results. For e.g. the study by Arisholm et al.[8] has used 38 class level metrics, while the study by Lu et al.[92] has used 62 class level metrics to measure different concepts of object oriented paradigm. Thus, in such cases, it was not possible to list all the metrics.

Table 3.2: Key Parameters Review

S.No.	**Study**	**Journal**	**Year**	**Metrics Used**	**Data Analysis Techniques Used**	**Performance Measures Used**	**Software Used**
1	[89]	Software - Practice and Experience	1998	NLOC	Boxplots	Median test	PMR system, implemented in C++, 2 releases R4 and R6 are considered.
2	[90]	Proceedings of the 6th International Software Metrics Symposium	1999	Studied the methods, attributes and inheritance relationships in a class	Boxplots	Median test	PMR system, implemented in C++, 2 releases R3 and R4 are considered.
3	[23]	IEEE International Conference on Software Maintenance	1999	No. of coupling metrics	LR	Hit (%)	LALO system (an open multi-agent system development environment), comprises of 90 classes.
4	[32]	Science of computer Programing	2002	CK [36]	Mean, standard deviation, Correlation coefficient	ANOVA	A system for decision making in telecommunications,written in C++, comprises 1044 classes.
5	[8]	IEEE Transactions on Software Engineering	2004	Various dynamic coupling measures (11), static coupling measures and size measures (27).	Multivariate regression	Goodness-of-fit of the model	Velocity: OSS, part of the Apache Jakarta Project, 4 sub releases (within one major release)are analyzed).

S.No.	Study	Journal	Year	Metrics Used	Data Analysis Techniques Used	Performance Measures Used	Software Used
6	[138]	IEEE Transactions on Software Engineering	2005	CBO, NOM , CK [36]	LR	Accuracy, False positive ratio, False negative ratio, Sensitivity, Goodness-of -fit	(1) JFlex: open source, lexical analyser generator for Java, latest release consists of 58 classes,13 subsequent releases are analysed, (2) JMol: used for displaying 3D images of chemical structures, latest release consists of 169 classes, 9 subsequent releases are analyzed.
7	[1]	Proceedings of the 32nd EUROMICRO Conference on Software Engineering and Advanced Applications	2006	No. of coupling metrics	J48, Jrip, PART, and, NBTree.	Accuracy	BOAP (program analysis toolbox system), written in Java and contains 394 classes.
8	[127]	11th European Conference on Software maintenance and Reengineering	2007	AID , ALD, LOC, MNOB, MPC, NIC, NOLV, NOP	Own proposed probabilistic approach	Accuracy, False positive ratio, False negative ratio, Sensitivity	JFlex: open source, lexical analyser generator for Java, latest release consists of 58 classes,14 subsequent releases are analysed.
9	[66]	Annual IEEE International Computer Software and Applications Conference	2008	BDM, CK [36]	Stepwise multiple regression	Goodness –of -fit	JFreechart: open source, Java class library for generating various types of charts.

S.No.	Study	Journal	Year	Metrics Used	Data Analysis Techniques Used	Performance Measures Used	Software Used
10	[128]	Journal of Software	2008	AID , ALD, LOC, MNOB, MPC, NIC, NOLV, NOP	Own proposed probabilistic approach	Accuracy, False positive ratio, False negative ratio, Sensitivity	JFlex : open source, lexical analyzer generator for Java, latest release consists of 58 classes.
11	[153]	IEEE Transactions on Software Engineering	2009	SLOC, NMIMP, NumPara (size metrics); 18 cohesion metrics, 20 coupling metrics, 17 inheritance metrics (OO metrics)	Linear regression equations	-	2 releases of Eclipse(open source development platform), written in Java. Eclipse 2.0 consists of 6751 Java files and Eclipse 2.1 consists of 7909 Java files.
12	[67]	Journal of Systems and Software	2010	CK [36], Lorenz and Kidd [91], MOOD [3], BDM	Stepwise multiple regression	Goodness of fit	JFlex : open source, lexical analyzer generator for Java, 9 releases are considered.
13	[50]	4th International Conference on Software Testing, Verification and Validation Workshops	2011	CK [36], QMOOD [9]	Predicted change prone classes using 'combined rank list' mechanism	Hit ratio (same as sensitivity), Cost ratio (ratio between cost of change prone classes and total change cost)	3 OSS: (1)JFreeChart (powerful open source charting library), (2)YARI (tool to debug, spy, spider, inspect and navigate), (3) UCDetector (Unnecessary Code Detector, an Eclipse plug-in to identify dead Java code).
14	[92]	Empirical Software Engineering	2011	62 OO metrics: 7 size , 18 cohesion , 20 coupling , and 17 inheritance	Random effect meta-analysis techniques	AUC (calculated using Wilcoxon test)	102 open source Java software systems, 2 releases are taken for each software.

S.No.	Study	Journal	Year	Metrics Used	Data Analysis Techniques Used	Performance Measures Used	Software Used
15	[124]	Technical report ISSN 1872-5392.	2011	CK [36], set of metrics to measure the complexity and the usage of interfaces, 2 metrics to measure external cohesion : interface usage cohesion, clustering metric	Correlation analysis, ML techniques (SVM, NB, ANN)	AUC, Precision, Recall	10 open source systems: 8 plugins from Eclipse, hibernate 2 and hibernate 3 systems.
16	[58]	Conference on Mining Software Repositories	2012	CK [36], 8 network centrality measures (e.g. nOutDegree, nInDegree, nPower etc.)	BN, ANN	AUC, Precision, Recall	19 plugin projects of the Eclipse platform and the Azureus project.
17	[45]	Journal of Software: Evolution and Process	2013	Derived evolutionary based metrics, CK [36]	LR	Correct classification rate (accuracy)	2 OSS: VSSPLUGIN1 and PeerSim developed using Java language.
18	[103]	International Journal of Machine Learning & Cybernetics	2013	CK [36], various other class level metrics	LR , bagging, RF, MLP	Sensitivity, Specificity, AUC	2 releases of 3 OSS developed in Java: (1)Frinika(complete music workstation software for various Operating System(OS), (2) FreeMind(mind mapper and hierarchical editor), (3)OrDrumbox(Java software drum machine and an audio sequencer).

S.No.	Study	Journal	Year	Metrics Used	Data Analysis Techniques Used	Performance Measures Used	Software Used
19	[105]	International Conference on Advances in Computing, Communications and Informatics	2013	CK[36], LOC	ANFIS, LR, Bagging, RF	Sensitivity, Specificity, AUC	2 OSS written in Java language: Frinika and CheckStyle
20	[104]	Proceedings of the 5th International Workshop on Emerging Trends in Software Metrics, 8-14	2014	CK [36]	GEP	Sensitivity, Specificity, Accuracy, Precision, F-measure, AUC	2 OSS: Simutrans and GlestBoth implemented in C++. Simutrans is a transport simulation game, Glest is a 3D strategy game and also an engine to make other strategy games.
21	[107]	16th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing	2014	NOM, DIT, RFC, NOC, CBO, TCC, LOC	Symbolic regression: an application of Genetic programming	Recall, Precision	4 evolving OSS developed in Java: ArgoUML, FindBugs, FOP, FreeCol

AID: Access of Imported Data, ALD: Access of Local Data, ANFIS: Adaptive Neuro-Fuzzy Inference System, LALO: Langage d'agents Logiciel Objet, MNOB: Maximum Number Of Branches, NBTree: NaiveBayes Decision Tree, NIC: Number of Imported Classes, NLOC: Number of uncommented Lines Of Code, NOLV: Number Of Local Variables, NOP: Number Of Parameters, PMR: Performance Management Traffic Recording

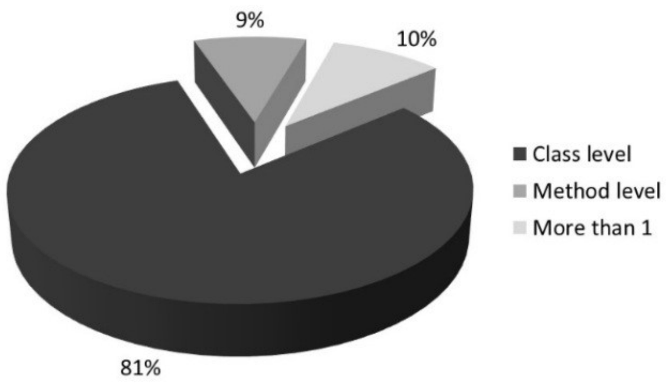


Figure 3.2: Distribution of Studies by Types of Metrics Used

Table 3.3: Types of Metrics and Software Repositories Used

Study	Types of Metrics	Types of Software Repositories
[89]	Class level	Commercial
[90]	Class level	Commercial
[23]	Class level	Open source
[32]	Class level	Unknown
[8]	Class level	Open Source
[138]	Class level	Open Source
[1]	Class level	Unknown
[127]	Method level	Open Source
[66]	Class level	Open Source
[128]	Method level	Open Source
[153]	Class level	Open Source
[67]	Class level	Open Source
[50]	Class level	Open Source
[92]	Class level	Open Source
[124]	Class level and Method level	Open Source
[58]	Class level and File level	Open Source
[45]	Class level	Open Source
[103]	Class level	Open Source
[105]	Class level	Open Source
[104]	Class level	Open Source

Study	Types of Metrics	Types of Software Repositories
[107]	Class level	Open Source

3.5.2 RQ2: What types of datasets are most widely used for prediction?

Datasets can be collected from various sources such as open source software, commercial/ proprietary software or academic/ university software. Open source datasets are mostly used by the researchers. Open source or public datasets should be widely used as software engineering can only be built using public datasets. Very few studies have used their own dataset i.e. commercial dataset. We have shown the distribution of studies according to the type of datasets used in figure 3.3. Studies have been divided according to the following classification:

- Public datasets: studies which have used only public or open source datasets.
- Private Datasets: studies which have used only private or proprietary or commercial datasets.
- Others: studies which have used both public and private datasets. This category also comprises of some studies which have not given any information about the type of dataset used.
- It can be seen from figure 3.3 that authors have predominantly used open source / public databases for their research.

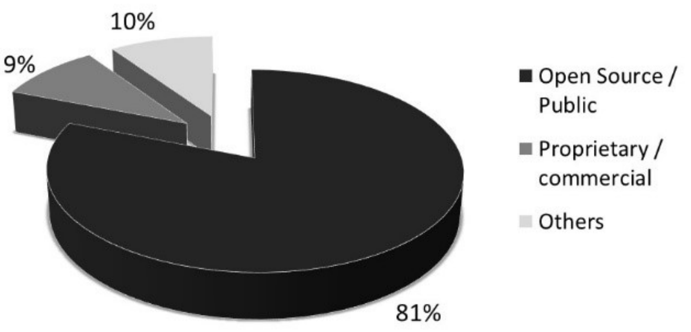


Figure 3.3: Distribution of Studies by Types of Software Repositories Used

3.5.3 RQ3: What type of machine learning techniques are used for model prediction?

As discussed, the methods used for predicting change prone classes can be broadly classified under two categories: ML and statistical. Table 3.3 lists all the methods used by the studies for change prediction under the column Data Analysis Techniques Used'. Some of the studies have neither used any ML technique, nor any statistical technique for the purpose of prediction. They have just done the analysis based on certain descriptive statistics such as mean, standard deviation etc. or have used the traditional graphical method such as boxplot. In other words, such studies have concluded their results based on only the descriptive statistics or boxplots. Thus, these methods are also listed in the table. Although nowadays, new ML techniques have replaced the traditional statistical methods, but it can be seen from table 3.3, that only few studies [1, 58, 103, 107, 124, 153] have used ML techniques for change prediction. Majority of the studies have used statistical methods and few have used both, statistical and ML techniques. Some of the studies have proposed their own ML or statistical models.

3.5.4 RQ4: What are the significant predictors of change proneness?

There is large number of independent variables used in literature for predicting change proneness. However, amongst a large set of independent variables used in a study, some of the variables are redundant and do not measure any useful information. Thus, such variables should be removed from the analysis before constructing the prediction models. As discussed in chapter 2, M.A. Hall [64] proved that "classification accuracy using reduced feature set is equal or better than accuracy using complete feature set." Thus, using various feature reduction techniques such as CFS and univariate LR; we should identify significant features or predictors. However, the literature shows that only two studies of change prediction [103, 153] have used feature reduction techniques. Thus, the literature does not allow us to conclude which variables are significant predictors of change proneness.

3.5.5 RQ5: Have appropriate performance measures used to evaluate the performance of the models?

There are number of OO metrics used for assessing the performance of different models. The performance of models greatly depends on how it is measured [8]. Researchers recommend the use of AUC as it is independent of the prior probabilities and the choice of arbitrarily cut-off value [68]. However, most of the studies have used traditional measures such as accuracy, sensitivity, specificity, and precision.

The main drawback of these measures is that they are calculated using some random cut-off value. The new measures such as balance, g-mean is unknown in the field of change prediction. Some studies have shown that g-mean is the best accuracy estimator of a prediction model [94]. Besides this, most of the studies have used the datasets where the number of change and non-change prone classes is not approximately the same (known as imbalanced data). To handle with imbalanced data some of the measures which are suggested as suitable parameters are precision, recall [62, 151], AUC [86, 110] and g- mean [72]. However, the researchers have not used appropriate performance measures to take into account imbalanced data.

3.5.6 RQ6: Have appropriate statistical tests used to measure the performance?

Once the performance of the models are evaluated using various measures, it is very important to use statistical tests to decide whether the differences between the performances of the models are real or random. There is lack of use of statistical tests followed by post hoc tests to predict change proneness. Chaumun et al. [32] used ANOVA to test the equality of change impact in three samples of a single dataset. The researchers generally adopt different statistical and common-sense techniques to decide whether the difference between the algorithms is actual or by-chance.

3.5.7 RQ7: What are the risk indicators for various OO metrics?

Once the metric values are calculated, there should be a technique to know if these values are good or bad. One of the techniques used is the identification of thresholds of metrics. Thresholds define an upper bound on the metric values such that the classes having metric values above the threshold values are risky. Thus, thresholds are also known as risk indicators. Knowing the potential classes at risk allow the managers in efficiently allocating the resources, allow designers to have a quick overlook on the metric value and redesign the classes if required and allow testers to prioritize their testing resources. Despite the potential advantages of using thresholds to predict quality attributes, there is lack of work on exploring the potential usage of thresholds to predict change prone classes. The threshold methodology has been explored in the field of fault prediction and there are few studies which used thresholds to predict fault prone classes [129, 131].

3.5.8 RQ8: What validation techniques are used to validate the models?

Once the models are constructed, it is very necessary to validate them to know the performance of the models. One of the popularly used technique is intra-project validation where the models are trained using the historical data (previous release) of some software and then it is validated on some future release of the same software. However, intra-project validation is not always possible because the historical data may not be available or does not exist. Thus, inter-project validation should be used where different projects are used for training and testing the model. Inter project validation also leads to more generalized results. The researchers have used inter project validation for predicting fault prone classes, but there is no study which has explored the use of inter-project validation for predicting change prone classes.

3.6 Review Conclusion and Future Directions

To assess the progress in the area of change prediction and to propose future guidelines, we have conducted a systematic review and studied the papers published in conference proceedings and journals of repute. We have provided an overview of the relevant studies in terms of some of the key parameters such as the metrics used, the data analysis methods used and the datasets used by each study. Besides this, important RQs are formulated and discussed which allowed to identify some gaps and gain significant insights from the existing studies. Further, some future guidelines or directions have been drawn which can be used by researchers and practitioners in their research. We discuss below the future guidelines with respect to each RQ:

- Metrics: The usage of method level metrics should be increased as the metrics at method level focus on more specific parts (i.e. methods). In other words, predicted modules are methods instead of classes. Therefore, more specific parts of source code (methods) which are change - prone can be identified. thus, their usage should be increased as they provide an insight at the low level of granularity.

- Software Repositories: Usage of commercial software repositories should also be increased as they are generally associated with real life applications.

- Data Analysis Techniques: Despite the importance and usage of ML in various areas such as finance, medical diagnosis, etc., there are very few studies analyzing the use of ML techniques. More studies need to be conducted in order to analyze the performance of ML techniques for prediction of quality attributes.

- Significant Predictors: More studies which use various data reduction techniques should be conducted to identify the variables which are significant predictors of change and fault proneness.

- Performance Measures: Studies using the appropriate performance measures such as AUC, G-mean and balance should be conducted so that performance is measured efficiently and imbalanced data is handled.

- Statistical Tests: The use of statistical tests should be increased to ensure that the performance of the models is not random.

- Risk Indicators: There are few studies that have established quality benchmarks of metrics known as thresholds for identification of fault prone classes. But there is lack of work exploring threshold methodology for change prediction. More studies should be done to identify threshold values of metrics for change prediction.

- Validation Techniques: To validate the results, cross project validation is being used by few fault proneness studies. But there is lack of work which has conducted cross project validation for change prediction. Studies using cross project validation to predict change proneness should be done.

Chapter 4

Investigation on Feasibility of Machine Learning Techniques for Predicting Software Change

4.1 Introduction

In this chapter, we explore the relationship between various OO metrics and change proneness to identify the metrics that are highly significant in predicting change prone classes. The selection of significant OO metrics has been done using univariate LR that is used to find the relationship between individual metric and change proneness. We have developed various models which can be used for predicting change prone classes. The models are validated with the help of various ML (AB, LB, RF, J48, BN, NB, bagging and MLP) and statistical (LR) techniques. The study has compared and analyzed the performance of LR technique with various ML techniques using performance measures such as sensitivity, specificity, precision and AUC. The results proved that ML techniques have comparable and competitive performance when compared with the statistical technique. For the purpose of empirical validation, an OSS, JTreeView written in Java is used.

The primary objective of this study is to focus and analyze the following broad perspectives:

a. Exploration of relationship between OO metrics and change proneness attribute of an OO software.

b. Evaluation of capability of ML techniques for prediction of change prone classes.

c. Comparison of predictive capability of ML techniques with the traditional statistical method, LR for detecting change prone classes.

d. Ascertaining the best ML technique for developing change prediction models.

The first perspective is motivated by the advantages of developing change prediction models. Identification of change prone classes would help in better resource planning as these classes should be allocated more resources so that defects and changes should not pass through to later stages of software development. Since change prone classes have higher probability of existence of defects, correction of these defects and identification of its sources is important. According to Pareto principle, 80% of changes in a software are concentrated in only 20% of its modules/classes [103]. More rigorous testing and verification activities should be performed on these few change prone classes in order to eliminate defects and improve customer satisfaction by delivering good quality products [124]. OO metrics, which categorize various structural aspects of a software like its size, cohesion, coupling etc. have been found to be good indicators of change prone nature of a class [45, 50, 53, 58, 92, 124]. This study validates the relationship between OO metrics to find out the best possible set of OO metrics which are efficient indicators of change using univariate LR.

The second perspective addresses the capability of ML techniques for prediction of change prone classes. Only few studies in past have used software metrics to ascertain the effectiveness of ML techniques for predicting change prone classes [58, 124]. Though, a number of studies have evaluated the effectiveness of ML techniques for developing defect prediction models [5, 60, 63, 79, 130, 134, 152], but the use of ML techniques to predict change prone classes is relatively new. Thus, the study aims to explore the ability of ML techniques for predicting change prone classes.

The third perspective attempts to compare the capability of ML techniques with the statistical technique, LR. Different techniques tend to give different results when data sets are varied. Moreover, few studies in literature [58, 124] have established the effectiveness of ML techniques for prediction of change prone classes but their comparison with the statistical method is not provided. Thus, it is important to compare the capability of ML techniques with the statistical methods so that researchers and practitioners can choose efficient techniques for change prediction tasks.

The last aim is to evaluate the best ML technique for change prediction tasks amongst the explored ML techniques in the study. The ranking is done on the basis of their predictive capability which is evaluated using sensitivity, specificity, precision and AUC performance metrics. The best ML technique can be used by different researchers and practitioners to identify change prone classes of a similar software.

This chapter is organized as follows: The research methodology is presented in section 4.2 which focuses on the software used and shows the descriptive statistics of all the independent variables. Section 4.3 shows the univariate and validation results.

The results of this chapter are published in [99].

4.2 Research Methodology

The basic methodology followed in this chapter is briefly explained: The first step involves the data collection process to retrieve the change data and the metrics of JTreeView. After the data is collected, univariate analysis is conducted. The significant metrics obtained after univariate analysis are used for model construction. In this study, multiple models are constructed using ML and statistical techniques. Finally, the performance of the models is evaluated using various performance measures. These steps are also depicted diagrammatically in figure 4.1. In this section, a brief description of software used for empirical validation is given followed by the empirical data collection process. We have also presented descriptive statistics of all the metrics used.

4.2.1 Software Used

We have used an OSS 'Java TreeView' (JTreeView) written in Java. It is a Cross-Platform Gene Expression Visualization Tool which is used for an interactive display of Clustered Gene Expression Data. In other words, JTreeView renders gene expression data into several interactive views. The source code of this OSS is available at http://sourceforge.net. Two releases are analyzed and change is calculated in the classes that are present in both the releases. We have considered the latest release i.e. 1.1.6 and one of the intermediate release i.e. 1.0.3. The release 1.1.6 contains 236 Java files and 710 classes whereas version 1.0.3 contains 173 files and 610 classes.

The latest release was last updated on 2011/06/13 and the release 1.0.3 was last updated on 2003/10/28.

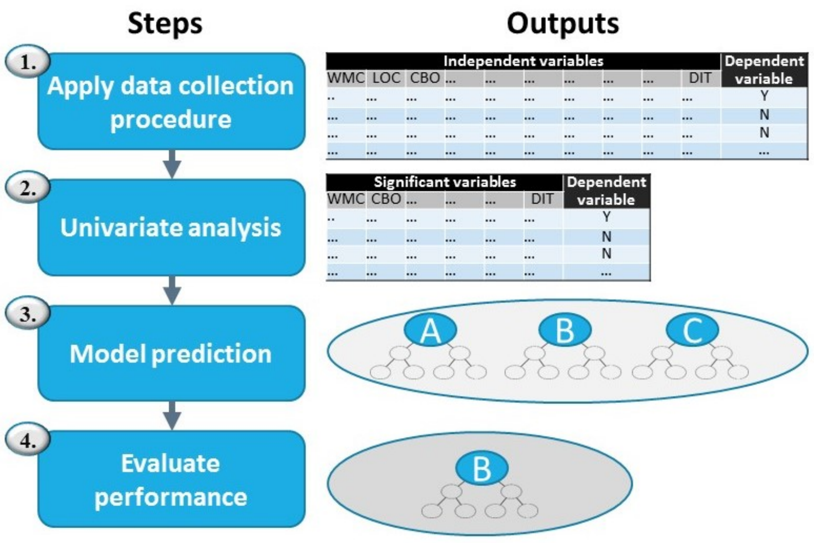


Figure 4.1: Primary Steps of Methodology

4.2.2 Empirical Data Collection

The change data of JTreeView is extracted from CVS software repository. CVS is a source control repository which keeps track of each and every change incurred as well as all the metadata regarding each change. The change collection process is explained in detail in chapter 2. In this section, we briefly summarize the entire process. The first step involves the extraction of the metrics collected using metric - collection open source tool known as 'Understand for Java'.

The next step is the collection of change data which is done using the CMS tool [96]. For the two releases under consideration, the change is calculated for classes that appeared in both the releases. Change is collected in terms of number of lines

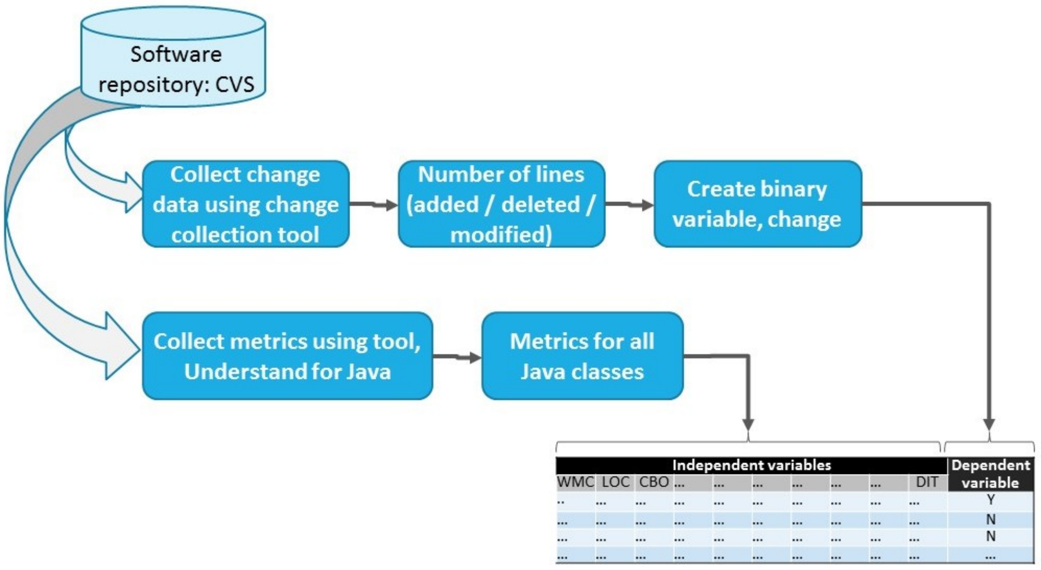

Software repository: CVS
Collect change data using change collection tool
Number of lines (added / deleted / modified)
Create binary variable, change
Collect metrics using tool, Understand for Java
Metrics for all Java classes
Independent variables
Dependent variable
WMC LOC CBO ... DIT
Y
N
N

Figure 4.2: Outline of Data Collection Process

added, deleted and modified in the classes of recent release with respect to the classes of the previous release. Finally, the report containing the software metrics and the change data is generated. This process is also explained diagrammatically in figure 4.2.

4.2.3 Descriptive Statistics

Various descriptive statistics are provided in table 4.1 which includes the minimum (Min.), maximum (Max.), mean (Mean) and standard deviation (SD) of all the metrics. The following important observations are obtained:

The size of a class is measured in terms of SLOC. It can be observed from the table that the range is between 0 to 100, concluding that JTreeView is a medium sized software. The mean values of NOC are 0.31 and the maximum value is 7, showing that the classes have very few children and use of inheritance is very less in the system.

Similar results are shown by other studies [30, 103, 134]. The LCOM measure has high maximum value (i.e. 100), showing that there is high cohesion in the system. Similar results have been shown by [103, 123].

Table 4.1: Descriptive statistics

S.No.	Metric	Min.	Max.	Mean	SD
1	NIB	1	6	1.33	0.7
2	CBO	0	23	2.34	4.27
3	NOC	0	7	0.31	1.05
4	NIM	0	62	7.44	10.17
5	NIV	0	31	3.45	4.97
6	NPRM	0	8	0.73	1.58
7	NPM	0	51	6.23	8.43
8	NPROM	0	9	0.42	1.46
9	RFC	0	88	11.25	17.23
10	DIT	1	4	1.65	0.65
11	LCOM	0	100	41.06	36.18
12	SLOC	2	853	91.65	130.91
13	BLOC	0	102	15.68	21.99
14	ELOC	0	511	48.69	75.13
15	ALOC	0	39	9.32	6.77
16	ALOCO	0	15	0.62	2.05
17	ACC	0	6	1.63	1.07

4.3 Result Analysis

In this section, the univariate and the validation results are presented.

4.3.1 Univariate Logistic Regression Analysis

Table 4.2 presents the results of univariate LR. The table provides the coefficient (β), standard error (SE), statistical significance (Sig.) and odds ratio (Exp(β)) statistic for each measure. The parameter 'Sig.' indicates whether each metric is significantly associated with change proneness at the significance level of 0.05. The metrics which are significant predictors of change proneness are shown in bold in table 4.2. Thus, only such metrics are used for model construction and the remaining are ignored. The coefficient 'β' shows the strength of the independent variable. The higher the value, the higher is the impact of the independent variable. The sign of the coefficient tells whether the impact is positive or negative.

Table 4.2: Univariate Analysis

S.No.	Metric	Sig.	β	SE	Exp(β)
1	NIB	0.126	0.603	0.394	1.828
2	**CBO**	0.003	0.431	0.147	1.539
3	NOC	0.075	0.896	0.503	2.451
4	**NIM**	0.002	0.148	0.048	1.159
5	**NIV**	0.02	0.146	0.063	1.157
6	**NPRM**	0.05	0.346	0.178	1.413
7	**NPM**	0.002	0.177	0.057	1.193
8	NPROM	0.142	1.305	0.889	3.687
9	**RFC**	0.003	0.135	0.046	1.145
10	DIT	0.082	0.59	0.082	1.803
11	**LCOM**	0.001	0.02	0.006	1.02
12	**SLOC**	0.009	0.009	0.003	1.009
13	**BLOC**	0.025	0.033	0.015	1.034
14	**ELOC**	0.011	0.015	0.006	1.015
15	ALOC	0.309	0.033	0.032	1.033
16	ALOCO	0.452	-0.08	0.106	0.923
17	ACC	0.8	-0.048	0.191	0.953

4.3.2 Validation Result Analysis

Table 4.3 shows the results of validation obtained on JTreeview software. Various ML and statistical models are constructed and their performance is evaluated using various parameters. To evaluate the performance of a model, it should be applied on a software which is different from the one one which it is trained. Thus, to divide the software into training and testing sets, K - cross validation is used with the value of K as 10 [44]. The diagrammatic representation of 10-cross validation is shown in figure 4.3. Table 4.3 shows the values of sensitivity, specificity, precision, and AUC for all the predicted models.

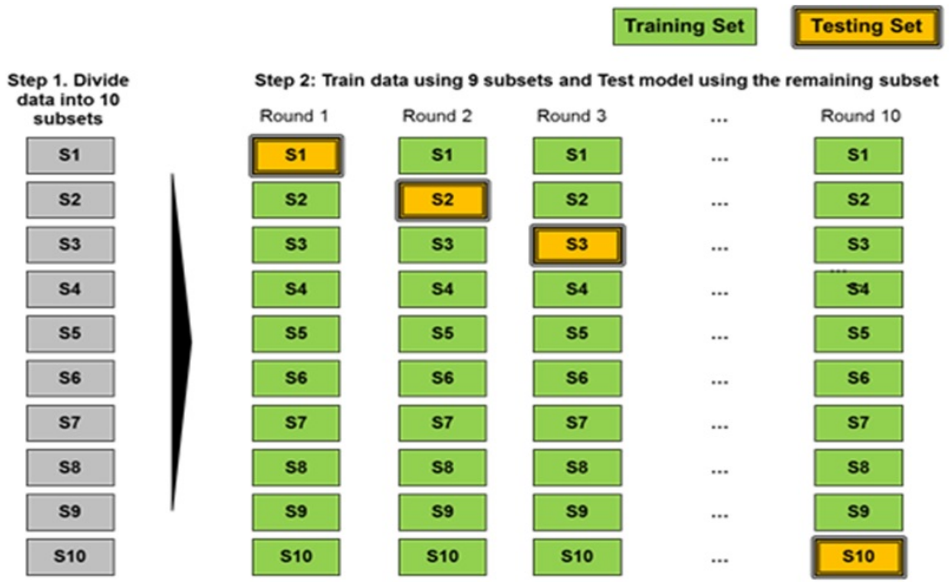

Training Set
Testing Set
Step 1. Divide data into 10 subsets
Step 2: Train data using 9 subsets and Test model using the remaining subset
Round 1
Round 2
Round 3
...
Round 10
S1
S2
S3
S4
S5
S6
S7
S8
S9
S10

Figure 4.3: 10 - Cross Validation

Table 4.3: Validation Results

S.No.	Techniques	Sensitivity	Specificity	Precision	AUC
1	AB	0.566	0.591	0.577	0.644
2	LB	0.698	0.659	0.68	0.717

S.No.	Techniques	Sensitivity	Specificity	Precision	AUC
3	RF	0.698	0.682	0.691	0.77
4	J48	0.698	0.659	0.68	0.687
5	BN	0.679	0.614	0.65	0.675
6	NB	0.66	0.75	0.701	0.719
7	Bagging	0.717	0.705	0.712	0.752
8	MLP	0.679	0.682	0.68	0.769
9	LR	0.679	0.682	0.68	0.768

Comparison among Various Machine Learning Models

Table 4.3 shows that among various ML models, RF has shown the best results in terms of AUC (0.770). The models predicted using bagging, LB, MLP and NB have also shown comparable AUC as that of RF with the values as 0.752, 0.720, 0.769, 0.719 respectively. When comparing the sensitivity and specificity values, highest values are shown by bagging which is 71.7% for sensitivity and 70.5% for specificity. The models predicted using J48 and LB have shown exactly the same value of sensitivity (69.8%) and specificity (65.9%) but at different cut-off points. This means, we can say that accuracy or precision of these two models is same (68.0%). When comparing the two boosting techniques i.e. AB and LB, there is significant difference in performance in terms of all the 3 measures – sensitivity, specificity and AUC. In fact, AB has shown the lowest values for sensitivity (56.6%), specificity (59.1%) and AUC (0.644). Thus, overall we concluded that among all the ML techniques, bagging has shown the best results with quite high values of sensitivity (71.7%), specificity (70.5%), precision (71.2%) and AUC (0.750). Although, RF has shown highest AUC, but its sensitivity and specificity values are quite low when compared with those of bagging. The ROC curves for all the models are shown in figure 4.4.

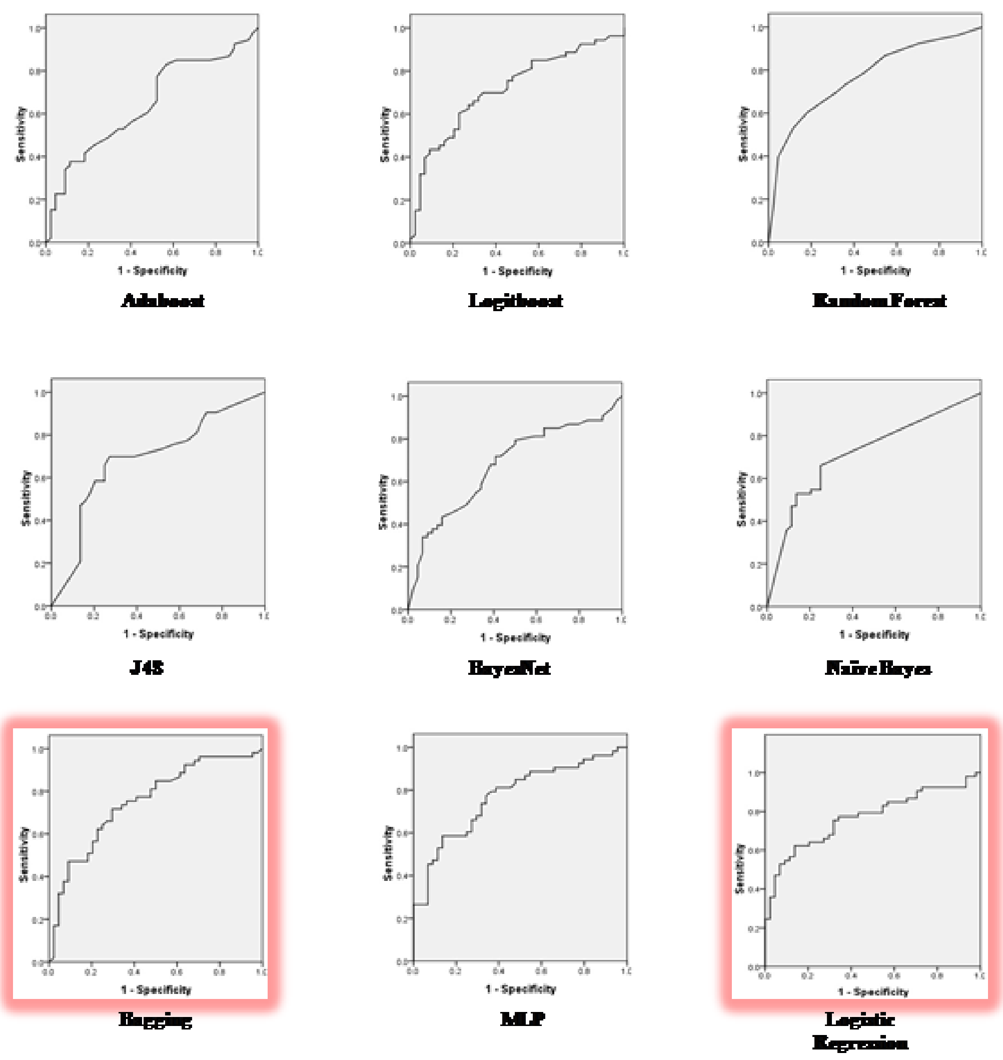


Figure 4.4: ROC Curves of Various Machine Learning and Statistical Models

Comparison between Machine Learning and Statistical Models

Although trend is shifting from the traditional statistical techniques to the modern ML techniques, but we observed that the statistical method (LR) has shown comparable results with the ML models. When compared with bagging, LR has shown better AUC (0.769), but sensitivity and specificity values are higher for bagging. In fact, the AUC for LR is higher than all the ML models except MLP and RF. Also, its sensitivity and specificity values are comparable with those of ML models. Hence, we concluded that LR has shown comparable performance with ML models.

4.4 Discussion

The broad focus of this study is to bring improvement in the quality of the software by predicting some parts of the software that are more prone to changes than others. We have used OO metrics for this purpose and have analyzed the relationship between metrics and change proneness. We have found significant relationship between most of the metrics and change proneness. Besides this, we have also developed models for improving software quality using various ML and statistical techniques. The performance of the ML models is compared with the performance of the model developed using the statistical technique. We observed that the statistical method has shown comparable and competitive performance with ML models. Among various ML techniques used, bagging has shown the best results with quite high values of sensitivity, specificity and AUC. Overall, we conclude that various quality models predicted can be used by researchers and practitioners in their studies to make further important conclusions.

Chapter 5

Models for Predicting Change Proneness for Popular Mobile Operating System, Android

5.1 Introduction

In literature, there are a number of ML techniques which have been used for constructing various prediction models. But, it has been observed that their performance varies across different software and thus, the conclusion regarding the superiority of one technique over the other cannot be made. Hence, it becomes necessary to conduct replicated studies in order to draw generalized conclusions which will allow us to establish well - formed theories. Besides this, several previous studies in software change prediction have used only a small number of ML techniques which does not provide an opportunity to fairly evaluate all the techniques. There are few studies of fault prediction [86] which have considered large number of techniques (22), but to the best of our knowledge, there is no change prediction study that has considered as many ML techniques. Thus, more studies need to be conducted which use a significant number of techniques such that a fair evaluation is possible. Further, there are different performance measures used across studies which may cause misleading results (particularly when the performance measures are based on the number of misclassified change or non - change prone classes). Also, as reported by Menzies et al. [110] and Myrtveit et al. [113], the studies conclude the results without testing for statistical significance. Finally, the past studies have validated the models on the same software from which they are derived. Thus, this chapter deals with four concerns: (i) no study to the best of our knowledge has used large number of data analysis techniques which allows for fair evaluation; (ii) studies in literature have not statistically compared the performance of techniques; (iii) incorrect use of performance measures across studies; (iv) validation of models from the same software from which they are trained. To address these concerns, we have used 15 data analysis techniques (14 ML techniques and a statistical technique) for predicting change prone classes and compared their performance. The comparison is based on AUC, g-mean and balance as they are most suitable for an unbalanced data. This extensive comparison allowed us to fairly evaluate most of the ML techniques and judge the performance of one technique over the other. Further, the performance of the models is also statistically validated using statistical tests. Finally, the models derived from a release ‘r’ are validated on the same release (using 10 - cross validation) and also on the releases ‘r+1’, ‘r+2’ (using inter - release validation). This application to a large extent increases the practical utility and relevance of change prediction models.

The following RQs are addressed in this work:

- **RQ1**: Among multiple ML techniques available, which technique can be used for predicting change prone classes of Android or similar dataset?
- **RQ2**: How does the performance of ML techniques compare with the statistical technique?
- **RQ3**: Which pairs of techniques (ML and statistical) are significantly different from each other for change prediction?
- **RQ4**: How do you compare the results of inter - release validation with 10 - cross validation?
- **RQ5**: Are the performance results consistent with different evaluation measures, viz. AUC, g-mean and balance?

The results are validated on five releases of an open source, widely used operating system in mobile phone and tablet computers, 'Android'. The results concluded that the ML techniques are effective in predicting change prone classes and thus, should be widely used by researchers and practitioners to reduce maintenance effort leading to efficient and better development of software.

This chapter is organized as follows. Section 5.2 focuses on the variables and the software used in this study. Various data analysis methods are discussed in section 5.3. Section 5.4 describes the performance measures used to evaluate the performance. The results of correlation analysis are also explained in this section. The results obtained using 10 - cross validation are summarized in section 5.5. Section

5.6 focuses on inter - release model prediction. Section 5.7 discusses various RQs stated in section 5.1. Finally, the results of this work are summarized in section 5.8.

5.2 Research Background

In this section, we give an overview of the variables (independent and dependent) and software used in this work.

5.2.1 The Variables Used

The work in this chapter is focusing on the use of various data analysis techniques for constructing models. The models are constructed with the help of various OO metrics including CK metrics [36] (the definition of each metric is given in chapter 2). The models are used for predicting change prone classes. Thus, the independent variables are various metrics and the binary dependent variable is change proneness.

5.2.2 Empirical Data Collection

We have used successive releases of an open source, Linux based operating system, Android. Android is a widely used operating system for mobile devices and developed by Google under the Apache license. Android has seen number of updates (either to fix bugs or to add new features) to its base operating system since its original release in September 2008. The first Android release was Android 0.9, followed by 1.0, then 1.1, 1.5 and so on. We analyzed the popularly and widely used releases of Android OS that are 2.3 (Gingerbread), 4.0 (Ice-cream Sandwich), 4.1 (Jellybean), 4.2 (Jellybean), 4.3 (Jellybean) and 4.4 (KitKat). We selected these releases as these are the stable releases used by the customers for a long period of time. We did not consider the releases before 2.3 because they did not stay in the market for long. Following Android 2.3 is the release 3.0 (code name Honeycomb) which we did not consider as it was used in tablets as an operating system and not used in non-tablet devices. Moreover, according to the general user's review, it was buggy and thus, rolled out quickly (http://www.quora.com/Why-is-Android-3-0-not available-on-smartphones) from the market. We calculated change between successive releases of Android. For example, change is calculated for the common classes between releases 2.3 to 4.0. Similarly, between the releases 4.0 to 4.1, 4.1 to 4.2, 4.2 to 4.3 and 4.3 to 4.4, the change is calculated. The change collection process is explained in detail in chapter 2. The source codes of all these releases are available at `http://source.android.com/source/initializing.html`. The details of each release which includes the total number of classes and the number of classes changed is given in chapter 2.

5.3 Data Analysis Methods

In this section, we discuss various ML techniques and statistical method used in this chapter.

5.3.1 Statistical Model

Logistic regression (LR) is one of the commonly used statistical modeling methods which is used in this work for model construction. We have also used univariate LR to find the metrics which are significant in predicting the dependent variable. The details about LR can be found in chapter 2.

5.3.2 Machine Learning Techniques

Table 5.1 presents the summary of 14 ML techniques used in this study. The use of large number of ML techniques allows their fair evaluation to judge the superiority of one technique over the other. We categorized the ML techniques under the following domains:

- Bayesian Learners (BL)
- Decision Trees (DT)
- Ensemble Learners (EL)
- Neural Networks (NN)
- Rule Based Learning (RBL)
- Support Vector Machines (SVM)
- Miscellaneous

We selected ML techniques in such a way that atleast one technique falls under each of the category. Besides this, the literature shows that there are various ML techniques which (such as Voted Perceptron (VP), Radial Basis Function (RBF), Alternating Decision Trees (ADT) etc.) have never been used for prediction of change prone classes. In this work, we choose some of these techniques so as to explore their usage in the field of change proneness prediction.

Table 5.1: Machine Learning Techniques Used

S.No.	Technique Used	Description	Category/Domain
1.	NaiveBayes (NB)	Naive Bayes is widely used inductive learning algorithm for ML and data mining. It is a simple probabilistic classifier that uses Bayes' theorem.	BL
2.	BayesNet (BN)	BN is also based on conditional probabilities and uses K2 as search algorithm [38].	BL
3.	J48	It is used for generating decision trees using C4.5 algorithm [122].	DT
4.	Alternating Decision Tree (ADT)	ADT is a combination of decision tree with effectiveness of boosting. The layers of the tree alternate between prediction nodes and decision nodes. The root node is a prediction node followed by decision nodes (decision tree stumps) in the next layer. The next layer then again consists of prediction nodes, and so on.	DT
5.	Bagging	Bagging is a voting method whereby different samples (training sets) are generated from a given sample by bootstrap.	EL
6.	Logitboost (LB)	LB is one of the popular releases of boosting. It uses a regression method as the base learner and it performs additive logistic regression [56].	EL
7.	Adaboost (AB)	AB combines a number of weak hypotheses to get better classification performance. For this, equal weights are assigned to all the training examples and then the weights of the incorrectly classified examples are increased on each round so that a weak learner is forced to focus on the hard examples in the training set [56].	EL
8.	Random Forest (RF)	Random forest is used for growing a number of classification trees, leading to a forest. To classify any new object, we input that object in each of the tree in the forest. Each tree gives a classification, and we say the tree ”votes” for that class [21].	EL
9.	Logistic Model Trees (LMT)	LMT is a classification tree with logistic regression function at the leaves [84]	EL
10.	Multilayer Perceptron (MLP)	MLP simulates biological neurons and model complex relationships between the input and the output with a set of hidden layers. It uses back propagation to learn instances.	NN

S.No.	Technique Used	Description	Category/Domain
11.	Radial Basis Function (RBF)	It is a neural network that uses normalized Gaussian radial basis function as activation function.	NN
12.	Decision Table Naive Bayes (DTNB)	DTNB is a hybrid of DT and NB which combines the advantages of both.	RBL
13.	Voted Perceptron (VP)	VP creates a hyperplane to distinguish between different types of input. It can be extended to non linear boundaries using kernel function. Polynomial kernel function is used for VP.	SVM
14.	Voting Feature Intervals (VFI)	VFI classifies using voting feature intervals which are constructed around each class for each attribute [40].	Misc.

5.4 Research Methodology

In this section, we describe the performance measures used for evaluating the models. The results of correlation analysis are also reported in this section.

5.4.1 Performance Evaluation

We need some parameters to measure the performance of the proposed models. In this chapter, we used three evaluation measures namely AUC, g-mean and balance. In all the five releases of Android which we have used, the number of change pone classes is very few as compared to non - change prone classes. Since the data used is imbalanced, we selected these measures. In addition to handling imbalanced data, AUC also deals with nosiy data, is independent of prior probabilites and is insensitive to the changes in the class distributions. Due to these reasons, it is widely used and recommended measure in various studies [31, 86, 110]. Besides this, some of the previous work has also shown that g-mean is the best accuracy estimator of a prediction model [94]. The description of each of these parameters is provided in chapter 2.

5.4.2 Validation Methods

There is a serious problem of overfitting, if the model is tested using the same software from which it is trained. In other words, the technique might perform well on the available data but poorly on future unseen test data. To overcome this problem of overfitting, we have used two validation techniques, K - cross validation and inter - release validation. For conducting inter - release validation, the models built using the OO metrics obtained from release ‘r’ of Android software is used to predict change prone classes of future releases, ‘r+1’. In other words, release ‘r’ is used as training set to construct the model and then the developed model is validated/tested on releases ‘r+1’, ‘r+2’, ‘r+3’ etc.

5.4.3 Correlation Analysis

We have conducted correlation analysis to study the variation among independent variables and the amount of correlation between them. Independent variables which are correlated with each other represent redundant information. Thus, it is very important to identify such correlated variables. The correlation coefficient in the correlation analysis determines the extent of correlation between different variables. The value of correlation coefficient ranges from -1 to +1. If there is a positive correlation between the two variables (metrics), it implies that the value of those two metrics increase together. Whereas, negative correlation implies that as one value increases, the other decreases. Since, the dependent variable in this study is categorical, we cannot check for the normality of the data. Hence, we calculated correlation using a non - parametric correlation test known as Spearman's correlation. For each release, tables 5.2, 5.3, 5.4, 5.5 and 5.6 show the Spearman's Rho coefficient of correlation between all the metrics. The studies in literature consider the correlations larger than 0.8 as high, in between 0.5 and 0.8 as moderate, and below 0.5 as low (Shatnawi and Wei 2008; Succi et al. 2005). In the tables, we have shown the correlation coefficients greater than 0.8 in bold. By definition of correlation, it is clear that correlation of a variable with itself will be totally perfect. In other words, the correlation coefficient value will be 1 (as shown by diagonal elements in the tables).

The table 5.2 (for release 2.3) shows that there are high correlations among the metrics NIM, NLM, NPM and WMC. Also, there is high correlation among LOC and WMC. Some of the metrics are moderately correlated with one another. For example, CBO shows moderate correlation with NIM, NIV, NLM, NPRM and WMC. For the release 4.0 (table 5.3), we can observe that NIM shows high correlation with NLM, WMC and NPM. Also, NLM shows high correlation with NPM and WMC. There is also a high correlation between LOC and WMC. Similarly, for the other releases, 4.1, 4.2 and 4.3 (tables 5.4, 5.5 and 5.6), there are few metrics that show high correlations among one another. These high correlations are undesirable and may cause collinearity as the metrics are not totally independent and represent redundant information. Thus, we have further calculated the condition number for the metrics. PC method is applied on the independent variables to find the maximum eigenvalue (e_max) and minimum eigenvalue (e_min). Then we calculated the conditional number given as $\lambda = \sqrt{e\_max/e\_min}$. If the value of conditional number is less than 30, we say that there does not exist multicollinearity between the variables [11]. Table 5.7 shows the condition number for all the metrics of each release. We can observe that the condition number is well below 30 for every release. This implies we can use all the metrics together in conjunction with one another. In other words, multicollinearity is tolerable.

Table 5.2: Correlation Analysis of Android 2.3

	CBO	NOC	NOM	NOA	NIM	NIV	NLM	RFC	NPRM	NPROM	NPM	LOC	DIT	WMC
NOC	0.11													
NOM	0.1	0.01												
NOA	0.29	0.02	0.51											
NIM	0.71	0.15	0.05	0.35										
NIV	0.69	0.07	0.02	0.25	0.73									
NLM	0.69	0.15	0.35	0.47	**0.96**	0.69								
RFC	0.29	0.04	0	0.07	0.32	0.28	0.3							
NPRM	0.64	0.02	0.19	0.3	0.71	0.65	0.73	0.26						
NPROM	0.42	0.21	0.02	0.21	0.56	0.38	0.53	0.33	0.31					
NPM	0.55	0.16	0.38	0.48	**0.86**	0.54	**0.92**	0.23	0.46	0.41				
LOC	0.77	0.07	0.18	0.43	0.78	0.74	0.78	0.24	0.7	0.39	0.64			
DIT	0.08	0	-0.13	-0.1	0.01	0	-0.03	0.55	0	0.12	-0.06	-0.03		
WMC	0.79	0.1	0.16	0.38	**0.85**	0.73	**0.85**	0.27	0.73	0.49	0.7	**0.91**	-0.02	
LCOM	0.35	0.08	0.18	0.31	0.32	0.32	0.35	0.06	0.25	0.15	0.33	0.32	-0.12	0.3

Table 5.3: Correlation Analysis of Android 4.0

	CBO	NOC	NOM	NOA	NIM	NIV	NLM	RFC	NPRM	NPROM	NPM	LOC	DIT	WMC
NOC	0.12													
NOM	0.11	0.01												
NOA	0.3	0.04	0.42											
NIM	0.71	0.2	0.09	0.35										
NIV	0.7	0.09	0.04	0.23	0.68									
NLM	0.7	0.19	0.34	0.44	**0.97**	0.65								
RFC	0.3	0.03	0	0.1	0.3	0.22	0.28							
NPRM	0.65	0.03	0.24	0.29	0.68	0.59	0.71	0.24						

	CBO	NOC	NOM	NOA	NIM	NIV	NLM	RFC	NPRM	NPROM	NPM	LOC	DIT	WMC
NPROM	0.42	0.26	0.03	0.22	0.58	0.34	0.56	0.29	0.31					
NPM	0.56	0.2	0.36	0.44	**0.88**	0.5	**0.92**	0.23	0.45	0.45				
LOC	0.78	0.09	0.19	0.45	0.76	0.67	0.76	0.27	0.67	0.4	0.64			
DIT	0.11	0	-0.12	-0.07	0.03	-0.01	0	0.51	0.02	0.12	-0.03	0.03		
WMC	0.36	0.06	0.18	0.29	0.32	0.32	0.35	0.09	0.27	0.14	0.33	0.35	-0.1	
LCOM	0.79	0.14	0.19	0.36	**0.84**	0.71	**0.84**	0.23	0.7	0.51	0.7	**0.88**	0	0.29

Table 5.4: Correlation Analysis of Android 4.1

	CBO	NOC	NOM	NOA	NIM	NIV	NLM	RFC	NPRM	NPROM	NPM	LOC	DIT	WMC
NOC	0.13													
NOM	0.14	0.02												
NOA	0.37	0.09	0.43											
NIM	0.74	0.24	0.13	0.45										
NIV	0.75	0.09	0.08	0.3	0.69									
NLM	0.73	0.23	0.35	0.52	**0.98**	0.67								
RFC	0.26	0.04	0	0.09	0.25	0.17	0.24							
NPRM	0.7	0.04	0.28	0.34	0.68	0.64	0.71	0.17						
NPROM	0.4	0.29	0.04	0.3	0.6	0.28	0.57	0.27	0.23					
NPM	0.59	0.25	0.35	0.52	**0.91**	0.52	**0.94**	0.22	0.49	0.52				
LOC	0.81	0.12	0.21	0.5	0.79	0.72	0.79	0.23	0.7	0.4	0.67			
DIT	0.07	-0.01	-0.12	-0.08	0	-0.03	-0.03	0.51	-0.02	0.11	-0.04	0.01		
WMC	0.36	0.07	0.19	0.29	0.32	0.32	0.34	0.07	0.29	0.14	0.32	0.34	-0.11	
LCOM	0.8	0.17	0.19	0.44	**0.86**	0.74	**0.86**	0.19	0.7	0.5	0.74	**0.89**	-0.02	0.28

Table 5.5: Correlation Analysis of Android 4.2

	CBO	NOC	NOM	NOA	NIM	NIV	NLM	RFC	NPRM	NPROM	NPM	LOC	DIT	WMC
NOC	0.12													
NOM	0.14	0.02												
NOA	0.32	0.06	0.44											
NIM	0.72	0.22	0.12	0.38										
NIV	0.72	0.09	0.07	0.25	0.67									
NLM	0.71	0.21	0.37	0.47	**0.97**	0.65								

	CBO	NOC	NOM	NOA	NIM	NIV	NLM	RFC	NPRM	NPROM	NPM	LOC	DIT	WMC
RFC	0.27	0.03	0	0.07	0.26	0.17	0.24							
NPRM	0.68	0.04	0.3	0.31	0.67	0.62	0.7	0.17						
NPROM	0.39	0.26	0.04	0.22	0.56	0.3	0.54	0.3	0.24					
NPM	0.57	0.22	0.37	0.48	**0.9**	0.49	**0.93**	0.21	0.47	0.47				
LOC	0.78	0.11	0.22	0.47	0.76	0.68	0.77	0.22	0.68	0.39	0.65			
DIT	0.09	-0.01	-0.11	-0.08	0.01	-0.03	-0.02	0.52	-0.01	0.13	-0.03	0.02		
WMC	0.36	0.06	0.19	0.3	0.32	0.32	0.35	0.09	0.28	0.15	0.33	0.35	-0.09	
LCOM	0.8	0.17	0.21	0.4	**0.86**	0.71	**0.86**	0.19	0.7	0.51	0.73	0.87	-0.02	0.29

Table 5.6: Correlation Analysis of Android 4.3

	CBO	NOC	NOM	NOA	NIM	NIV	NLM	RFC	NPRM	NPROM	NPM	LOC	DIT	WMC
NOC	0.12													
NOM	0.11	0.02												
NOA	0.28	0.06	0.48											
NIM	0.69	0.23	0.1	0.35										
NIV	0.69	0.1	0.03	0.21	0.62									
NLM	0.68	0.22	0.37	0.46	**0.96**	0.59								
RFC	0.27	0.03	0	0.08	0.26	0.18	0.24							
NPRM	0.65	0.05	0.28	0.28	0.6	0.53	0.64	0.18						
NPROM	0.39	0.27	0.03	0.21	0.57	0.29	0.54	0.29	0.24					
NPM	0.54	0.23	0.38	0.48	**0.89**	0.46	**0.94**	0.2	0.44	0.46				
LOC	0.77	0.11	0.2	0.43	0.75	0.66	0.75	0.23	0.68	0.39	0.63			
DIT	0.09	-0.01	-0.1	-0.07	0.01	-0.02	-0.02	0.52	0	0.13	-0.03	0.02		
WMC	0.38	0.06	0.19	0.28	0.34	0.35	0.37	0.09	0.32	0.15	0.33	0.36	-0.08	
LCOM	0.77	0.18	0.19	0.37	0.85	0.67	0.85	0.19	0.69	0.51	0.72	0.86	-0.01	0.3

5.5 Result Analysis

This section shows the results of various data analysis (ML and statistical) techniques used for predicting change proneness using OO metrics. Before constructing various

ML models, we obtained a subset of good features (independent variables that are correlated with the dependent variable, but not related with each other) using CFS [111]. M.A. Hall proved that "classification accuracy using reduced feature set is equal or better than accuracy using complete feature set." [64]. The advantages of using CFS are efficient and more accurate model prediction and reduction in dimensionality. The independent variables obtained after applying CFS are used in the construction of ML models. Whereas, the statistical model is constructed using the subset of metrics obtained by univariate LR. The metrics of each release obtained after applying CFS and univariate LR are listed in table 5.7. In this section, we report the results obtained of 10 - cross and inter - release validation.

Table 5.7: CFS and Univariate LR Results

Android Release	Metrics Selected after CFS	Metrics Selected after Univariate LR
Android 2.3	CBO, NOA, NIV, NLM, RFC, LOC, DIT, WMC, LCOM	CBO, NOC, NOM, NOA, NIM, NIV, NLM, NPRM, NPROM, NPM, LOC, DIT, WMC, LCOM
Android 4.0	CBO, NOA, NIV, RFC, LCOM, WMC	CBO, NOM, NOA, NIM, NIV, NLM, RFC, NPRM, NPROM, NPM, LOC, DIT, WMC, LCOM
Android 4.1	CBO, NOC, NOA, NIV, RFC, LOC	CBO, NOC, NOA, NIM, NIV, NLM, NPRM, NPM, LOC, DIT, WMC, LCOM
Android 4.2	CBO, NOA, NIV, NLM, RFC, NPROM, LOC, DIT, WMC	CBO, NOM, NOA, NIM, NIV, NLM, NPRM, NPROM, NPM, LOC, DIT, WMC, LCOM
Android 4.3	CBO, NOA, NIV, NLM, WMC	CBO, NOM, NOA, NIM, NIV, NLM, NPRM, NPROM, NPM, LOC, DIT, WMC, LCOM

5.5.1 Model Evaluation Using 10 - Cross Validation

The accuracy of the models predicted would be highly optimistic if testing is done on the same software from which training model is derived. Thus, we have used 10 -

cross validation to ensure that the training and testing sets belong to different parts of a given release. Tables 5.8, 5.9 and 5.10 report the results of 10 - cross validation of 15 data analysis techniques on five releases of Android. Table 5.8 shows the AUC, table 5.9 shows the g-mean and table 5.10 shows the balance. For each release, the highest AUC, g-mean and balance are shown in bold. We observe that for all the releases, bagging and RF have shown the best performance as compared to other techniques. For example, AUC of bagging is highest for all the releases. It is equal to or more than 0.75 for all the releases except Android 4.3. Besides this, bagging has also shown highest g-mean and balance for majority (4 out of 5) of the releases. RF follows bagging in all the evaluation measures viz. AUC, g-mean and balance. In fact, we can observe that the performance of bagging and RF is quite similar in most of the cases. For example, RF has shown the same g-mean (i.e. highest value) as bagging for 3 out of 5 releases. Also, it has given AUC equal to or more than 0.75 for all the releases except Android 4.3. Besides bagging and RF, other techniques such as ADT and BN have AUC greater than 0.7 corresponding to most of the releases of the Android. Amongst remaining techniques, majority of them have AUC more than 0.65 for most of the releases. ADT and BN have also shown high values of g-mean and balance. For all the performance measures, VFI and VP have shown the worst performance with the lowest values of AUC, g-mean and balance. When comparing the performance of statistical model with ML models, we observe that the performance of LR is comparable to the performance of ML models. LR model has shown comparable values of AUC, g-mean and balance for all the releases. We summarize the results as follows: bagging has outperformed all the other techniques, followed by RF, ADT and BN in the given order. Their performance is consistent across all the releases and for all the three performance measures. Such a consistent result across all the releases and for all the performance measures provide us with a very strong reason to use bagging to predict change prone classes of any future release of Android. Thus, we would suggest practitioners and researchers to use bagging to identify change prone classes of Android or any other similar software.

Table 5.8: 10 - Cross Results of AUC as Performance Measure

Technique	2.3	4	4.1	4.2	4.3	Average
NB	0.70	0.64	0.62	0.68	0.68	0.664
BN	0.72	0.70	0.70	0.69	0.69	0.700
J48	0.72	0.68	0.70	0.69	0.64	0.686
ADT	0.72	0.70	0.71	0.70	0.69	0.704
Bag	**0.77**	**0.75**	**0.77**	**0.76**	**0.73**	0.756
LB	0.72	0.70	0.69	0.69	0.68	0.696
AB	0.70	0.67	0.69	0.69	0.67	0.684
RF	0.76	**0.75**	0.76	0.75	0.72	0.748
LMT	0.74	0.69	0.73	0.68	0.68	0.704
MLP	0.72	0.68	0.68	0.69	0.67	0.688
RBF	0.70	0.65	0.61	0.65	0.67	0.656
DTNB	0.73	0.69	0.71	0.68	0.68	0.698
VP	0.61	0.55	0.52	0.53	0.53	0.548
VFI	0.64	0.62	0.65	0.62	0.62	0.630
LR	0.71	0.66	0.66	0.69	0.69	0.682

Table 5.9: 10 - Cross Results of G-mean as Performance Measure

Technique	2.3	4	4.1	4.2	4.3	Average
NB	63.9	51.8	46.8	48.7	47.7	51.78
BN	65.5	56.1	51.5	50.7	49.8	54.72
J48	67.2	56.2	52.0	51.8	47.5	54.94
ADT	64.9	56.0	53.1	51.4	51.4	55.36
Bag	**69.1**	**61.0**	**57.2**	**56.3**	**52.9**	59.30
LB	65.4	56.5	51.7	51.2	50.0	54.96

Technique	2.3	4	4.1	4.2	4.3	Average
AB	64.0	53.8	50.4	50.9	48.2	53.46
RF	68.8	**61.0**	57.1	55.6	52.4	58.98
LMT	66.5	55.9	56.3	49.2	48.1	55.20
MLP	65.0	55.4	50.5	50.3	48.3	53.90
RBF	63.8	53.1	48.2	48.2	50.3	52.72
DTNB	66.0	55.2	52.7	50.4	49.0	54.66
VP	59.4	53.8	48.5	54.5	50.0	53.24
VFI	60.5	50.3	48.3	45.4	45.7	50.04
LR	63.6	53.9	50.0	50.8	48.8	53.42

Table 5.10: 10 - Cross Results of Balance as Performance Measure

Technique	2.3	4	4.1	4.2	4.3	Average
NB	0.65	0.60	0.59	0.62	0.63	0.618
BN	0.67	0.65	0.64	0.64	0.65	0.650
J48	0.68	0.65	0.65	0.65	0.61	0.648
ADT	0.66	0.65	0.66	0.65	0.66	0.656
Bag	**0.71**	**0.70**	0.70	**0.70**	**0.69**	0.700
LB	0.67	0.65	0.65	0.65	0.65	0.654
AB	0.65	0.62	0.63	0.65	0.63	0.636
RF	0.70	0.69	**0.71**	0.7	0.68	0.696
LMT	0.68	0.64	0.70	0.63	0.63	0.656
MLP	0.66	0.64	0.63	0.64	0.63	0.640
RBF	0.65	0.61	0.58	0.61	0.63	0.616
DTNB	0.67	0.64	0.66	0.63	0.64	0.648
VP	0.61	0.41	0.31	0.33	0.33	0.398
VFI	0.65	0.61	0.58	0.61	0.63	0.616
LR	0.62	0.58	0.61	0.58	0.59	0.596

Statistical Tests

From the above discussion, we concluded that there is difference in the performance of various techniques and bagging outperformed all the other techniques. We have also statistically evaluated the results using Friedman test. This will confirm that the performance difference is not random and will evaluate the superiority of one technique over the other techniques. Friedman test is applied on the results of all the three evaluation measures. The Friedman test resulted in a p-value of 0.000 for all the evaluation measures. The p-value of 0.000 proved that the results are significant at the 0.05 level of significance over 14 degrees of freedom. Since, the p-value is less than the significance level (0.05), the null hypothesis is rejected and alternate hypothesis is accepted. Thus, we say that there is statistical difference in the performance of participant techniques. The rank obtained by each technique when using AUC, g-mean and balance is shown in table 5.11. The rank demonstrates the performance of all the techniques (lowest numerical rank value shows the highest performance). From table 5.11, we observe that the top rank (shown in bold) is obtained by bagging, followed by RF and ADT for all the evaluation measures. In other words, bagging, RF and ADT have shown consistent performance. We concluded that the performance difference is not random as bagging and RF have also statistically shown superior performance over the other techniques. Thus, bagging and RF can be used for predicting change prone classes. The techniques such as RBF, VFI,VP have shown the worst performance.

Table 5.11: Ranks Obtained by Friedman Test

Technique	Ranks when AUC used	Ranks when g-mean used	Ranks when balance used
NB	11.3	13.2	11.9
BN	5.6	7	6.2
J48	8.3	6.2	6.4
ADT	4.5	5.6	5
Bag	**1.1**	**1.4**	**1.4**

Technique	Ranks when AUC used	Ranks when g-mean used	Ranks when balance used
LB	7.6	5.8	5.2
AB	9.6	9.9	9.3
RF	1.9	2	1.7
LMT	6.2	7.6	6.9
MLP	8.6	9.2	8.9
RBF	12.4	11.6	12.1
DTNB	6.7	7.4	7.1
VP	15	8.5	15
VFI	13.6	14.4	13.6
LR	8.5	10.2	9.3

Since the null hypothesis is rejected, we also performed a post - hoc analysis to examine the statistical difference between the pairs of different techniques using Nemenyi test (explained in chapter 2). In Nemenyi test, we construct null and alternate hypothesis for each pair. The total possible pairs with 15 techniques are 15C2 which equals 105. Thus, there are 105 sets of null and alternate hypotheses. Since it is not possible to state all the hypotheses, we state some of the hypotheses where the comparison is with bagging:

- Ho1: The performance of bagging and ADT techniques do not differ significantly.
- Ha1: The performance of bagging and ADT techniques differ significantly.
- Ho2: The performance of bagging and RF techniques do not differ significantly.
- Ha2: The performance of bagging and RF techniques differ significantly.
- Ho3: The performance of bagging and NB techniques do not differ significantly.

- Ha3: The performance of bagging and NB techniques differ significantly.

The results of pair-wise comparisons of the techniques when the evaluation measure is AUC, g-mean and balance are shown in tables 5.12, 5.13 and 5.14 respectively. Each cell shows the difference in the ranks of two corresponding techniques. We compared the value of this rank difference with the CD to accept or reject the null hypothesis. The CD is calculated as follows:
In this study, k = 15 and n =5. The value of q_a for 15 techniques at 'a' = 0.05 is 4.796/ $\sqrt{2}$=3.392.
Thus, CD= $q_a\sqrt{\frac{k(k+1)}{6n}} = 3.392\sqrt{\frac{15(15+1)}{6.5}} = 9.594$

From the tables, we observe that for some of the pairs, the rank differences (shown in bold) are more than the CD. For such pairs, we rejected the null hypothesis and concluded that the performance of the corresponding two techniques differ significantly. For other pairs, where rank difference is less than the CD, we accepted the null hypothesis and concluded that there is no statistical difference in the performance of the two techniques. We observe that bagging has always shown significantly different performance from some of the ML techniques when either of the evaluation measure is used. For example, when g-mean is used, it has shown significantly different performance from NB and RBF. Whereas, when AUC and balance are used, it has shown significantly different performance from four ML techniques (NB, RBF, VFI and VP). Besides bagging, few other ML techniques have also shown significantly different performance from other techniques. Thus, although the result of Friedman test says that there is significant difference in the performance of participant techniques, but the pairwise comparison in Nemenyi test says that there are only few techniques that have shown significant difference in their performance as compared to other techniques. From the result of Nemenyi test as well, we again concluded that bagging should be used by researchers and practitioners for predicting change prone classes. Overall, we concluded that out of 15 ML techniques used in this study, bagging should be used to predict change prone classes in any future release of Android or any other similar project because of the following 3 reasons:

1. Bagging has outperformed all the other techniques in terms of all the three evaluation measures, i.e. AUC, g-mean and balance.
2. In all the releases as well, i.e. Android 2.3, 4.0, 4.1, 4.2 and 4.3, bagging has shown the best performance as compare to other techniques. In other words, bagging has shown consistent performance across all the releases.
3. The result of Nemenyi test shows that bagging is the only method that has shown significantly different performance from most of the other ML techniques.

Table 5.12: Results of Nemenyi Test when AUC is Used

Technique	BN	J48	ADT	Bag	LB	AB	RF	LMT	MLP	RBF	DTNB	VP	VFI	LR
NB	5.7	3	6.8	**10.2**	3.7	1.7	9.4	5.1	2.7	1.1	4.6	3.7	2.3	2.8
BN	0	2.7	1.1	4.5	2	4	3.7	0.6	3	6.8	1.1	9.4	8	2.9
J48		0	3.8	7.2	0.7	1.3	6.4	2.1	0.3	4.1	1.6	6.7	5.3	0.2
ADT			0	3.4	3.1	5.1	2.6	1.7	4.1	7.9	2.2	**10.5**	9.1	4
Bag				0	6.5	8.5	0.8	5.1	7.5	**11.3**	5.6	**13.9**	**12.5**	7.4
LB					0	2	5.7	1.4	1	4.8	0.9	7.4	6	0.9
AB						0	7.7	3.4	1	2.8	2.9	5.4	4	1.1
RF							0	4.3	6.7	**10.5**	4.8	**13.1**	**11.7**	6.6
LMT								0	2.4	6.2	0.5	8.8	7.4	2.3

Technique	BN	J48	ADT	Bag	LB	AB	RF	LMT	MLP	RBF	DTNB	VP	VFI	LR
MLP									0	3.8	1.9	6.4	5	0.1
RBF										0	5.7	2.6	1.2	3.9
DTNB											0	8.3	6.9	1.8
VP												0	1.4	6.5
VFI													0	5.1

Table 5.13: Results of Nemenyi Test when G-mean is Used

Technique	BN	J48	ADT	Bag	LB	AB	RF	LMT	MLP	RBF	DTNB	VP	VFI	LR
NB	6.2	7	7.6	**11.8**	7.4	3.3	**11.2**	5.6	4	1.6	5.8	4.7	1.2	3
BN	0	0.8	1.4	5.6	1.2	2.9	5	0.6	2.2	4.6	0.4	1.5	7.4	3.2
J48		0	0.6	4.8	0.4	3.7	4.2	1.4	3	5.4	1.2	2.3	8.2	4
ADT			0	4.2	0.2	4.3	3.6	2	3.6	6	1.8	2.9	8.8	4.6
Bag				0	4.4	8.5	0.6	6.2	7.8	**10.2**	6	7.1	13	8.8
LB					0	4.1	3.8	1.8	3.4	5.8	1.6	2.7	8.6	4.4
AB						0	7.9	2.3	0.7	1.7	2.5	1.4	4.5	0.3
RF							0	5.6	7.2	**9.6**	5.4	6.5	**12.4**	8.2
LMT								0	1.6	4	0.2	0.9	6.8	2.6
MLP									0	2.4	1.8	0.7	5.2	1
RBF										0	4.2	3.1	2.8	1.4
DTNB											0	1.1	7	2.8
VP												0	5.9	1.7
VFI													0	4.2

Table 5.14: Results of Nemenyi Test when Balance is Used

Technique	BN	J48	ADT	Bag	LB	AB	RF	LMT	MLP	RBF	DTNB	VP	VFI	LR
NB	5.7	5.5	6.9	**10.5**	6.7	2.6	**10.2**	5	3	0.2	4.8	3.1	1.7	2.6
BN	0	0.2	1.2	4.8	1	3.1	4.5	0.7	2.7	5.9	0.9	8.8	7.4	3.1
J48		0	1.4	5	1.2	2.9	4.7	0.5	2.5	5.7	0.7	8.6	7.2	2.9
ADT			0	3.6	0.2	4.3	3.3	1.9	3.9	7.1	2.1	**10**	8.6	4.3
Bag				0	3.8	7.9	0.3	5.5	7.5	**10.7**	5.7	**13.6**	**12.2**	7.9
LB					0	4.1	3.5	1.7	3.7	6.9	1.9	**9.8**	8.4	4.1
AB						0	7.6	2.4	0.4	2.8	2.2	5.7	4.3	0

Technique	BN	J48	ADT	Bag	LB	AB	RF	LMT	MLP	RBF	DTNB	VP	VFI	LR
RF							0	5.2	7.2	**10.4**	5.4	**13.3**	**11.9**	7.6
LMT								0	2	5.2	0.2	8.1	6.7	2.4
MLP									0	3.2	1.8	6.1	4.7	0.4
RBF										0	5	2.9	1.5	2.8
DTNB											0	7.9	6.5	2.2
VP												0	1.4	5.7
VFI													0	4.3

5.6 Inter - Release Model Prediction

We also evaluated the accuracy of the models on successive releases of Android. For this, we used the prediction models of Android 2.3 to predict the change prone classes in the future releases, i.e. Android 4.0, 4.1, 4.2 and 4.3. Table 5.15 shows the result when prediction models of Android 2.3 are applied to Android 4.0 and 4.1, whereas table 5.16 shows the result when Android 2.3 prediction models are applied to Android 4.2 and 4.3. We observe that bagging has shown the highest AUC, g-mean and balance when we are validating Android 4.1 using Android 2.3. Similar results are obtained when Android 4.2 and 4.3 are validated with Android 2.3. When we are validating Android 4.0, we observe that bagging has shown the highest value of AUC and balance but not the g-mean. The highest g-mean is given by NB and bagging shows the second highest value. Overall, the results of all the inter - release validation also show that bagging has outperformed the other techniques. This result is consistent with the result of 10 - cross validation. We summarize three common trends observed in the results:

1. First, for all the inter release validations, bagging gives the highest perfor-

mance. This consistent result across all the inter - release validations can be used by researchers for conducting inter - release model prediction.

2. Second, the prediction results are consistent as the system is evolving. In other words, the performance of predicting change prone classes in Android 4.0, 4.1, 4.2 and 4.3 using Android 2.3 is similar in terms of AUC, g-mean and balance. This suggests that the researchers can expect a similar performance when predicting change prone classes of any future release using the metric prediction models of Android 2.3.

3. Third, the results of inter - release validation of Android 2.3 on Android 4.0, 4.1, 4.2 and 4.3 are comparative and competitive to the results of validation of the releases itself (i.e. using 10 - cross validation). In most of the cases, the results of inter - release validation are even better than 10 - cross validation. This has also been shown diagrammatically in figure 5.1. Figure 5.1 shows the bar chart depicting the average AUC, g-mean and balance of each release for both 10 - cross validation and inter - release validation. We observe that the average balance of all the releases is higher for inter - release validation as compared to 10 - cross validation. Similarly, the average AUC and g-mean of releases 4.3 and 4.0 respectively is higher for inter - release validation as compared to 10 - cross validation. This concludes that inter - release validation is useful and effective in accurate prediction of change prone classes.

Table 5.15: Validating Android 2.3 on 4.0 and 4.1

	Android 2.3 on Android 4.0			Android 2.3 on Android 4.1		
Technique	AUC	g-mean	balance	AUC	g-mean	balance
NB	0.71	**63.6**	0.65	0.66	48.41	0.61
BN	0.67	54.0	0.62	**0.73**	**66.44**	**0.68**
J48	0.65	55.7	0.64	0.63	50.57	0.63
ADT	0.69	54.2	0.63	0.68	51.86	0.64
Bag	**0.72**	58.9	**0.68**	0.71	52.32	0.66
LB	0.68	54.4	0.63	0.69	51.35	0.64
AB	0.67	53.5	0.62	0.67	50.24	0.63
RF	**0.72**	58.1	0.67	0.71	52.66	0.66
LMT	0.68	56.2	0.65	0.68	52.08	0.65
MLP	0.70	57.7	0.66	0.69	51.84	0.65
RBF	0.66	53.2	0.61	0.66	48.70	0.61
DTNB	0.68	54.8	0.63	0.69	51.05	0.64
VP	0.59	50.5	0.58	0.60	46.72	0.59
VFI	0.62	50.9	0.59	0.62	50.50	0.61
LR	0.68	55.1	0.63	0.66	48.69	0.61

Table 5.16: Validating Android 2.3 on 4.2 and 4.3

	Android 2.3 on Android 4.2			Android 2.3 on Android 4.3		
Technique	AUC	g-mean	balance	AUC	g-mean	balance
NB	0.68	49.44	0.63	0.69	49.11	0.64
BN	0.68	50.50	0.64	0.69	50.90	0.64
J48	0.64	49.92	0.63	0.64	49.18	0.64
ADT	0.68	50.75	0.63	0.69	49.39	0.64
Bag	**0.71**	**52.26**	**0.66**	**0.72**	**51.95**	**0.67**
LB	0.69	50.44	0.63	0.70	49.53	0.64
AB	0.68	51.04	0.63	0.69	49.48	0.64
RF	0.70	51.30	0.65	**0.72**	51.51	**0.67**
LMT	0.65	49.13	0.63	0.67	49.68	0.64
MLP	0.70	51.77	**0.66**	0.71	50.81	0.66
RBF	0.68	49.71	0.63	0.69	48.02	0.63

	Android 2.3 on Android 4.2			Android 2.3 on Android 4.3		
Technique	AUC	g-mean	balance	AUC	g-mean	balance
DTNB	0.69	50.43	0.64	0.69	49.03	0.64
VP	0.61	46.98	0.60	0.62	48.60	0.61
VFI	0.62	46.40	0.59	0.62	45.52	0.60
LR	0.68	50.25	0.64	0.69	49.10	0.64

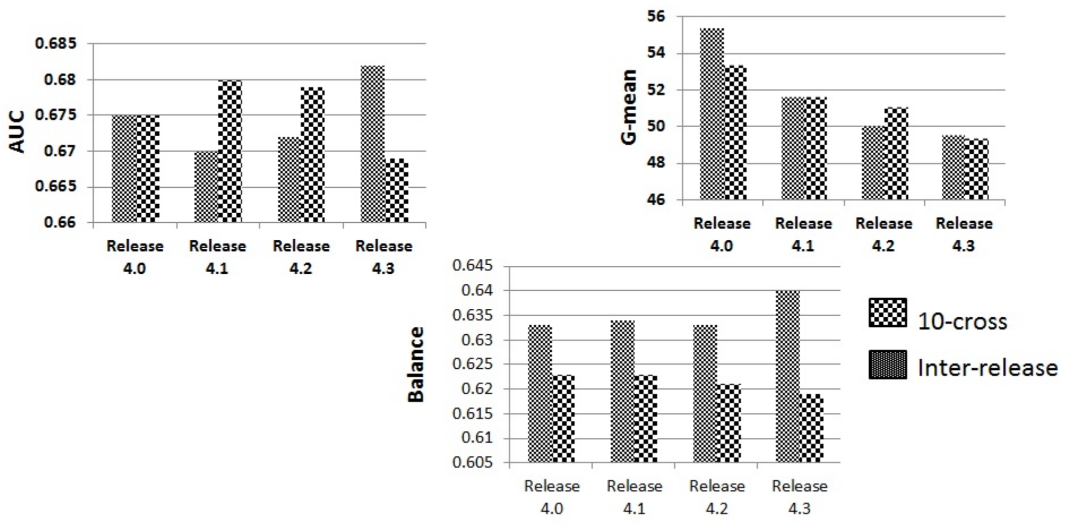


Figure 5.1: Comparison between 10 - Cross and Inter - Release Validation

5.7 Research Question Analysis

In this section, we discuss various RQs stated by us in section 5.1. We have provided answers to each one of these RQs in various sections of this chapter. Here, we try to summarize them.

- RQ1. Among multiple ML techniques available, which technique can be used for predicting change prone classes of Android or similar dataset?

Among multiple ML techniques available, the results of 10 - cross validation indicated that the performance of bagging is best in terms of AUC, g-mean and balance. This has also been statistically evaluated using Friedman test. Bagging has shown the highest rank when either of the evaluation parameter is used. Among the remaining techniques, RF follows bagging by showing the second best performance. Thus, bagging and RF can be used for predicting change prone classes of any future release of Android or of any similar dataset.

- RQ2. How does the performance of ML techniques compare with the statistical technique?

  The statistical technique (LR) used in this study has shown comparable and competitive results with ML techniques for both 10 - cross and inter - release validation.

- RQ3. Which pairs of techniques (ML and statistical) are significantly different from each other for change prediction.

  Among a total of 105 pairs possible for each of the evaluation measure, only few pairs have shown significantly different performance. The number of pairs which have shown significantly different performance when AUC, g-mean and balance is used are 8, 5 and 10 respectively (shown in bold in tables 5.12, 5.13, 5.14). The results show that bagging has always shown significantly different performance than some of the ML techniques in terms of all the evaluation measures.

- RQ4. How do you compare the results of inter - release validation with 10 - cross validation?

The results of inter - release validation are comparable (and even better in some cases) to the results when the models are validated using the same dataset (i.e. using 10 - cross validation). Thus, researchers can carry out inter - release validation to predict change prone classes. In other words, they can use a previously available release to predict the change prone classes in any future release.

- RQ5. Are the performance results consistent with different evaluation measures, viz. AUC, g-mean and balance?

  Yes, the results are consistent when different evaluation measures are used. For all the evaluation measures, bagging has shown the best performance followed by RF.

5.8 Discussion

We have investigated the performance of 15 different data analysis techniques for correct and accurate prediction of change prone classes in the earlier phases of SDLC. For this purpose, we have used various OO metrics and investigated the use of these metrics in the prediction of change proneness in five releases of Android, 2.3, 4.0, 4.1, 4.2 and 4.3. The performance of different techniques is evaluated using AUC, g-mean and balance. We have also performed inter - release validation by using the prediction models of Android 2.3 to predict the change-prone classes in the future releases, i.e. Android 4.0, 4.1, 4.2 and 4.3. We found the consistent results across all the inter release validations. The important findings of the work are:

1. Among multiple ML techniques used, bagging has outperformed all the other

techniques. The results of the Friedman test also shows the same result, i.e. superiority of bagging for change prediction.

2. The performance of statistical technique is comparable to the performance of ML techniques. Thus, researchers can also use the statistical technique for prediction of change proneness.

3. The statistical Nemenyi test shows very few pairs to be statistically different in terms of their performance. The number of pairs showing significantly different performance is 8, 5 and 10 when AUC, g-mean and balance are used respectively.

4. The results of inter - release validation are comparable to the 10 - cross validation. This confirms the generalizability of the models developed for change prediction in this study.

Chapter 6

Fault Prediction Considering Threshold Effects of Object Oriented Metrics

6.1 Introduction

The design of software can be assessed in the industry by identifying the out of range values of various OO metrics. The software design measurements can be used by the software industry in multiple ways: (1) Software designers can use them to obtain quality benchmarks to assess and compare various software products [4]; (2) Managers can use the metrics in controlling and auditing the quality of the software during the SDLC [129, 153]; (3) Software developers can use the metrics to identify problematic areas and use source code refactoring to improve the internal quality of the software [16, 54, 133]; (4) Software testers can use the metrics in effectively planning and allocation of testing and maintenance resources [4, 81, 128]. The focus of this chapter is to study the effect of thresholds of OO metrics on fault proneness. In other words, we have calculated the thresholds of OO metrics to predict fault prone of various OSS. For prediction of fault prone classes, many researchers have suggested various metrics models [22, 24, 28, 53, 63, 79, 92, 118, 152]. The use of these metric models in the early phases of SDLC allow effective allocation of maintenance resources and thus, leads to reduction of the maintenance cost. However, building and running metric based models on a daily basis is a time consuming task and thus, it is impractical [131]. A more effective way is to define some meaningful thresholds for all the OO metrics. A threshold of a metric is that value of the metric above which a class is considered to be risky and hence require careful attention and scrutiny. Chidamber et al. [34] said that one of the important uses of OO metrics is to identify extreme values (threshold values) of these metrics. The classes above these extreme values will have higher complexity and require management attention. The developers may use thresholds for various purposes: (1) to identify refactoring candidates such as bad code smells [129], (2) to predict the existence of bugs in the software [131] and (3) to identify design anomalies. Besides these, there are a number of advantages of using thresholds as stated in chapter 2.

To calculate the threshold values, we have assessed the use of a methodology proposed by Bender [12] based on LR. We calculated the thresholds at different risk levels and validated the models using various ML techniques (BN, NB, RF, SVM, MLP). We constructed binary models using the threshold values of OO metrics and compared thier accuracy with the corresponding non - binary models. The empirical validation is carried out on proprietary NASA software, KC1 (Metrics data repository: http://www.mdp.ivv.nasa.gov 2000), implemented in C++ programming language and two open source Promise software, Apache IVY and JEdit. To demonstrate the effectiveness of the proposed methodology, inter - project validation has also been done on three OSS, Apache Ant and Apache Tomcat and Sakura.

This chapter is organized as follows: Section 6.2 explains the background of the research, i.e. the dependent and independent variables used along with description of the datasets. Section 6.3 explains the basic methodology followed in this study. The concept and usage of 'threshold values' is also explained in this section. Next, section 6.4 summarizes the univariate and threshold analysis. The results of validation are discussed in section 6.5, along with the results of inter - project validation. Finally, the results of the work are summarized in section 6.6.

The results of this chapter are published in [100].

6.2 Research Background

In this section, we present the independent and the dependent variables used in this chapter. The details of the software along with their descriptive statistics is also provided.

6.2.1 Dependent and Independent Variables

In this study, the popular CK metrics [36] are used as independent variables. In addition to CK metrics, LOC metric is also included as it is the most common metric to measure size of the software. The binary dependent variable in this chapter is fault proneness.

6.2.2 Empirical Data Collection

We have used various software from Promise data repository, namely KC1, Ivy, Tomcat, Ant, and JEdit. Besides these software, Sakura editor is also used. KC1 is a proprietary software obtained from the NASA Metrics Data Program (NASA 2004, PROMISE), whereas Ivy, Tomcat, Ant, JEdit and Sakura are OSS. To show the effectiveness of the proposed methodology, generalization and comparison of results is very important which is conducted using the OSS. KC1 is implemented in C++ programming language and consists of 145 classes, 2107 methods and 40K lines of code. It consists of both the class level and the method level static metrics. We have used KC1 for class level metrics. The system consists of seven class level metrics including six CK metrics. The information present in the error reports is used to collect the fault data from KC1. From the error report, we found that the total number of faults are 669, which also included the faults that had "not a fault" keyword. But, we considered only those faults that are either produced from the source code or the design and removed all the faults that had "not a fault" keyword used as the reason of closure of the error report. This reduced the total number of faults to 642. These faults are spread across 59 classes out of a total of 145 classes. Thus, KC1 consists of 59 faulty classes.

Apache Ant is a Java library and command-line tool for automating software processes. The main usage of Ant is building of Java applications. Ant can also be used effectively to build non Java applications, for instance C or C++ applications (ant.apache.org). Apache Ivy is a powerful dependency manager that manages dependencies of any kind. It is integrated with Apache Ant and thus follows same design principles as Apache Ant (ant.apache.org/ivy). Apache Tomcat is a software implementation of Java Servlet and the JavaServer Pages (JSP) specifications from Sun Microsystems. It provides a "pure Java" HTTP web server environment to run a Java code (tomcat.apache.org). JEdit and Sakura, both are text editors for programmers. Sakura is available under the GNU General Public License 2.0 and implemented in Java programming language, whereas Sakura is available under Shareware license and is implemented in C++. All these datasets (except Sakura) are from Promise data repository collected by Jureczko and Madeyski [77], and Jureczko and Spinellis [78] using CKJM tool. The data for Sakura is collected using metric tool family, Understand for C and Understand for Java [145]. The summary of the details of OSS, which include the version of the software used, the number of instances (data points) and the number of faulty instances in the software is provided in chapter 2. Figure 6.1 indicates the percentage of classes which are found to be actually faulty in each of the five OSS.

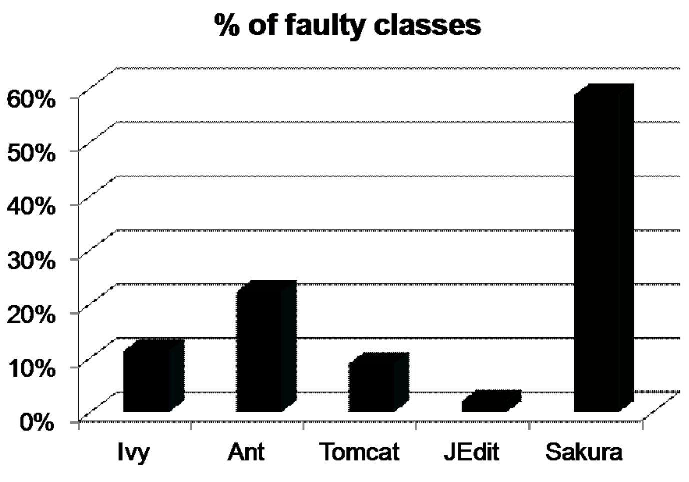


Figure 6.1: Summary of Faulty Classes

6.2.3 Descriptive Statistics

We calculated various statistics including minimum (Min.), maximum (Max.), mean (Mean), median (Median), standard deviation (SD) and the percentiles (25%ile and 75%ile) for the metrics of all the six software. Table 6.1 shows the descriptive statistics for KC1.

Table 6.1: Descriptive Statistics of KC1

Metric	Min.	Max.	Mean	Median	SD	25th %ile	75th %ile
WMC	0	100	17.42	12	17.45	8	235.5
DIT	0	6	1	1	1.26	0	1.5
NOC	0	5	0.21	0	0.7	0	0
CBO	0	24	8.32	8	6.38	3	14
RFC	0	222	34.38	28	36.2	10	44.5
LCOM	0	100	68.72	84	36.89	56.5	96
LOC	0	2313	211.25	108	345.55	8	235.5

We can make the following important observations regarding some of the metrics [134]:

- LOC: The size of a class is measured in terms of LOC. We can see that the minimum value for LOC is 0 while the maximum is 2313, which is a not a very large value.
- NOC and DIT: The value of NOC is 0 in 75% of the classes. This shows that the number of immediate children in 75% of the classes is 0, which concludes low inheritance in the system. This is also shown by DIT, the maximum value of DIT is 6. Also, 75% of the classes do not have even 2 levels of inheritance.
- LCOM: Cohesion in the system is not much high as shown by LCOM with the

maximum value as 100 (which is quite less).

Tables 6.2, 6.3, 6.4, 6.5 and 6.6 show the descriptive statistics of Ivy, Ant, Tomcat, JEdit and Sakura respectively.

Table 6.2: Descriptive Statistics of Ivy

Metric	Min.	Max.	Mean	Median	SD	25th %ile	75th %ile
WMC	1	157	11.28	6	15.15	3	13
DIT	1	6	1.79	1	1.25	1	2
NOC	0	17	0.37	0	1.32	0	0
CBO	1	150	13.23	8	16.57	5	16
RFC	1	312	34.04	19	44.68	6	40
LCOM	0	11794	131.58	6	712.19	0	45.75
LOC	1	2894	249.34	85.5	428.26	20	267

Table 6.3: Descriptive Statistics of Ant

Metric	Min.	Max.	Mean	Median	SD	25th %ile	75th %ile
WMC	0	120	11.07	7	11.98	4	14
DIT	1	7	2.52	2	1.4	1	4
NOC	0	102	0.73	0	4.8	0	0
CBO	0	499	11.05	6	26.34	4	11
RFC	0	288	34.36	23	36.03	11	43
LCOM	0	6692	89.15	6	349.94	0	53
LOC	0	4541	280.07	143	411.87	52	325

Table 6.4: Descriptive Statistics of Tomcat

Metric	Min.	Max.	Mean	Median	SD	25th %ile	75th %ile
WMC	0	252	12.96	7	18.62	3	14
DIT	1	6	1.69	1	1.05	1	2
NOC	0	31	0.36	0	1.97	0	0
CBO	0	109	7.57	4	11.1	2	9
RFC	0	511	33.47	17	44.98	7	40

Metric	Min.	Max.	Mean	Median	SD	25th %ile	75th %ile
LCOM	0	29258	176.28	4	1159.19	0	42
LOC	0	7956	350.44	112	644.84	25	373.5

Table 6.5: Descriptive Statistics of JEdit

Metric	Min.	Max.	Mean	Median	SD	25th %ile	75th %ile
WMC	0	351	12.35	7	24.51	3	13
DIT	1	8	2.37	1.5	1.93	1	3
NOC	0	38	0.45	0	2.43	0	0
CBO	0	346	14.32	8.5	25	4	15
RFC	0	540	39.85	24.5	56.34	9	52.75
LCOM	0	41713	259.91	6	2184.69	1	36
LOC	1	12535	411.31	176	946.96	41.25	451

Table 6.6: Descriptive Statistics of Sakura

Metric	Min.	Max.	Mean	Median	SD	25th %ile	75th %ile
WMC	2	275	17.19	9	32.78	6	14.75
DIT	0	2	0.43	0	0.59	0	1
NOC	0	14	0.38	0	1.69	0	0
CBO	0	64	5.33	2	9.4	1	6
RFC	2	275	25.35	15	33.21	8.25	35
LCOM	0	94	56.13	60	25.56	42	75
LOC	13	15227	681.5	162.5	1873.67	66.75	351.25

Following important observations can be made from these tables:

- LOC: The size of the class measured in terms of lines of code ranges from 1 to 2894 (Ivy), 0 to 4541 (Ant) and 0 to 7956 (Tomcat). This shows that while Tomcat and Ant are medium sized software, Ivy is smaller software. But both JEdit and Sakura are large sized software with the LOC ranging from 1 to 12535 for JEdit and 1 to 15227 for Sakura.

- NOC and DIT: The mean values of NOC for all the software are quite low. The mean values of NOC for Ivy, Ant, Tomcat, JEdit and Sakura are 0.37, 0.73, 0.36, 0.45 and 0.38 respectively. These low values indicate that there are very few children and inheritance is not much used in the systems. Similar results are shown by KC1.

- LCOM: In contrast to the degree of cohesion in KC1, the maximum value of LCOM is 11794 in Ivy, 6692 in Ant and 29258 in Tomcat. This indicates that there is high cohesion in the systems.

6.3 Research Methodology

The basic approach followed in this work is shown in figure 6.2. Each of these steps has been explained in detail in the subsequent sections. To begin with, we conducted univariate analysis to determine the metrics that are significant in predicting fault proneness. Once the significant metrics are determined, threshold values are calculated for these metrics. Validation of these threshold values are conducted using various ML techniques. Finally, the validation is carried on the same project for which thresholds are calculated as well as on the other projects.

6.3.1 Calculation of Threshold Values

This section explains the methodology used to compute the threshold values. We have used the Bender method [12] known as Value of an Acceptable Risk Level (VARL) to compute the threshold values where the acceptable risk level is given by a probability

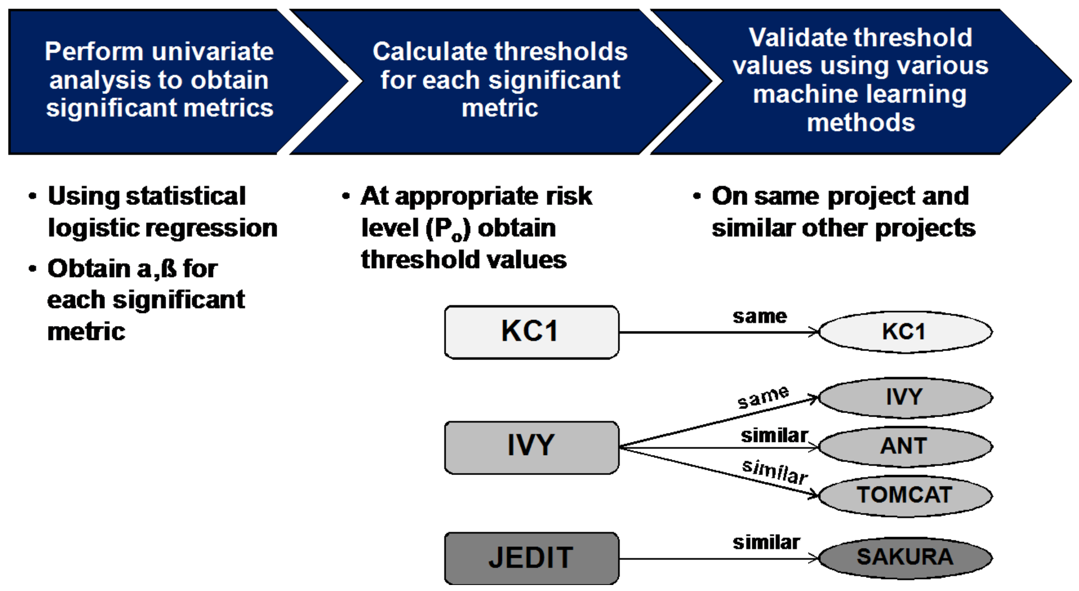

Perform univariate analysis to obtain significant metrics
Calculate thresholds for each significant metric
Validate threshold values using various machine learning methods
• Using statistical logistic regression
• Obtain a,ß for each significant metric
• At appropriate risk level (P_o) obtain threshold values
• On same project and similar other projects
KC1
same
KC1
IVY
same
IVY
similar
ANT
similar
TOMCAT
JEDIT
similar
SAKURA

Figure 6.2: Proposed Methodology

Po (e.g. Po = 0.05 or 0.01). For the classes with metrics values below VARL, the risk of fault occurrence is lower than the probability, Po. In other words, Bender [12] has suggested that the value of Po can be any probability which can be considered as the acceptable risk level. We have calculated the results of VARL at six levels of risk (between Po= 0.01 and Po= 0.15) and selected the most appropriate risk level.

The detailed description of VARL is given in [12]. The formula for VARL is given as follows:

VARL = $\frac{1}{\beta}[ln[\frac{Po}{1-Po}] - \alpha]$

Where, α = constant; β = estimated coefficient; Po = acceptable risk level

In this formula, α and β are obtained using univariate LR. The example below shows the calculation of the threshold value of CBO metric of Ivy software. Referring table 6.8 (univariate analysis for Ivy) to obtain the values of α and β, we get α = -2.598 and β = 0.033.

VARL = $\frac{1}{\beta}[ln[\frac{Po}{1-Po}] - \alpha]$

= $\frac{1}{0.033}[ln[\frac{Po}{1-Po}] - (-2.598)]$ (the values of Po are taken between 0.01 to 0.15)

= $\frac{1}{0.033}[ln[\frac{0.07}{1-0.07}] - (-2.598)]$ (taking Po=0.07)

= 0.34

Thus, we say VARL for CBO metric at risk level of 0.07 is 0.34

6.4 Result Analysis

In this section, the results obtained of univariate and threshold analysis are explained. Also, the results of validating the threshold values are presented in this section.

6.4.1 Univariate Analysis

Threshold values are calculated for the metrics of KC1, Ivy and JEdit. Therefore, we conducted univariate analysis for these software to find the effect of individual metrics on fault proneness. Tables 6.7, 6.8 and 6.9 show the coefficient (β), constant (α), and statistical significance (Sig.) for each metric of KC1, Ivy and JEdit. The 'Sig.' parameter tells about the association between each metric and fault proneness. If the 'Sig.' value is below or at the significance threshold of 0.05, then the metric is said to be significant in predicting fault proneness. We can observe that for both the software (KC1 and Ivy), LCOM, NOC, and DIT are not significant and hence are not used for further analysis.For JEdit, NOC and DIT are found to be insignificant. The significant metrics are shown in bold in the tables.

Table 6.7: Univariate Analysis of KC1

Metric	Sig.	β	α
WMC	0.021	0.028	-0.886
DIT	0.5	0.09	-0.497
NOC	0.213	-0.401	-0.332
CBO	0	0.207	-2.238
RFC	0.037	0.011	-0.785
LCOM	0.796	0.001	-0.488
LOC	0	0.005	-1.353

Table 6.8: Univariate Analysis of Ivy

Metric	Sig.	β	α
WMC	0	0.05	-2.786
DIT	0.818	-0.032	-1.997
NOC	0.46	-0.149	-2.011
CBO	0	0.033	-2.598
RFC	0	0.023	-3.131
LCOM	0.108	0	-2.126
LOC	0	0.002	-2.874

Table 6.9: Univariate Analysis of JEdit

Metric	Sig.	β	α
WMC	0.001	0.041	-4.486
DIT	0.696	0.061	-4.007
NOC	0.598	-0.376	-3.788
CBO	0.012	0.023	-4.259
RFC	0	0.016	-4.698
LCOM	0.003	0.001	-4.066
LOC	0.021	0.001	-4.137

6.4.2 Threshold Analysis

We have calculated the threshold values for the metrics that are found to be significant. The threshold values are calculated at different Po values (between 0.01 and 0.15). As we have discussed, Po is defined as the acceptable risk level, i.e. risk level that can be acceptable to classify a class as non - faulty. Thus, low values of Po are desirable. The lowest value of Po using which we get the threshold values within the observation range (i.e. positive values) of all the metrics is considered to be the best value of Po. The threshold values at different values of Po for all the significant metrics are shown in tables 6.10, 6.11 and 6.12 for KC1, Ivy and JEdit. We observe that the threshold values of all the metrics change significantly as the value of Po changes. This shows, Po plays a significant role in calculating threshold values.

Analysis for KC1

Table 6.10 shows the results of threshold analysis for KC1. The results show that at all levels of risks except at Po = 0.15, the VARL values of metrics are out of range (i.e. negative values) for some of the metrics. For example, the threshold values at Po = 0.05, 0.08 and 0.1 are outside the observation range of RFC and WMC metrics, whereas threshold values at Po = 0.01 lay outside the range of RFC, WMC and LOC metrics. Thus, the thresholds at these levels of risks cannot be considered and hence are ignored. The only risk level which gives the threshold values within the observation range of all the metrics is 0.l5. Hence, Po = 0.15 is said to be the most appropriate risk level for KC1. In other words, for the classes with metric values below VARL, the risk of fault occurrence is lower than the probability of 15%. The threshold values of CBO, RFC, WMC and LOC at Po = 0.15 are 7.17, 2.88, 4.74 and 119.93 respectively.

Table 6.10: Thresholds at Different Risk Levels for KC1

Metric	VARL at 0.01	VARL at 0.05	VARL at 0.06	VARL at 0.08	VARL at 0.1	VARL at 0.15
WMC	-39.63	-14.03	-11.03	-6.24	-2.44	4.74
CBO	1.17	4.63	5.04	5.69	6.2	7.17
RFC	-110.06	-44.89	-37.27	-25.06	-15.39	2.88
LOC	-128.53	14.85	31.6	58.46	79.75	119.93

Analysis for Ivy

Table 6.11 shows the threshold values for the metrics of Ivy at different risk levels. The results show that the threshold values at Po $\geq$ 0.07 are within the observation range of all the metrics. Since Po is the acceptable risk level, we consider the lowest value i.e. 0.07 as the appropriate risk level. The threshold values at 0.07 risk level are 3.99 for WMC, 0.34 for CBO, 23.67 for RFC and 143.66 for LOC. These values are validated on the same software as well as on two different OSS, Ant and Tomcat.

Table 6.11: Thresholds at Different Risk Levels for Ivy

Metric	VARL at 0.01	VARL at 0.05	VARL at 0.06	VARL at 0.07	VARL at 0.08	VARL at 0.1	VARL at 0.15
WMC	-36.18	-3.17	0.69	3.99	6.87	11.78	21.03
CBO	-60.52	-10.5	-4.65	0.34	4.72	12.14	26.16
RFC	-63.66	8.11	16.5	23.67	29.94	40.6	60.71
LOC	-860.56	-35.22	61.23	143.66	215.83	338.39	569.7

Analysis for JEdit

The values of threshold for all the significant metrics of JEdit at different risk levels are shown in table 6.12. We observe that among all the risk levels between 0.01 and 0.15, the risk level 0.02 gives the threshold values within the range, i.e. positive values. This means, the classes whose metrics values are below their threshold values, the risk of fault occurrence is lower than 2 %. The threshold values at 0.02 risk level are 14.49, 15.96, 50.39, 174.18 and 245.18 for the metrics WMC, CBO, RFC, LCOM, and LOC respectively.

Table 6.12: Thresholds at Different Risk Levels for JEdit

Metric	VARL at 0.01	VARL at 0.015	VARL at 0.02	VARL at 0.05	VARL at 0.06	VARL at 0.08	VARL at 0.1	VARL at 0.15
WMC	-2.66	7.35	14.49	37.6	42.3	49.84	55.82	67.11
CBO	-14.61	3.24	15.96	57.15	65.54	78.98	89.64	109.76
RFC	6.43	32.09	50.39	109.6	121.65	140.98	156.3	185.21
LCOM	-529.12	-118.6	174.18	1121.56	1314.47	1623.65	1868.78	2331.4
LOC	-458.12	-47.59	245.18	1192.56	1385.47	1694.65	1939.78	2402.4

6.5 Results Discussion

We calculated the accuracy of the models before and after converting the data into binary. The metrics of software are converted to binary using the threshold values of the metrics of the same software or some other similar software. If the value of a metric is more than its threshold value, then the metric value is changed to 1 indicating high risk and changed to 0 otherwise, indicating low risk. For example, the threshold values are calculated for one proprietary software (KC1) and two OSS (Ivy and JEdit).

The threshold values of KC1 and Ivy are validated on the same software (i.e. KC1 and Ivy respectively) and other similar software. The threshold values of Ivy are validated on two different OSS, Ant and Tomcat. Similarly, the threshold values of JEdit are validated on Sakura. In other words, we have conducted inter - project validation on three software (Ant, Tomcat and Sakura). This allows to prove the effectiveness of proposed methodology and give generalized results. Inter - project validation using the proprietary software, KC1 is very difficult because of the two reasons: (1) we do not have access to different proprietary data and (2) the results of proprietary software cannot be compared with that of OSS as their building methodology and environment are very different. The results of validation are shown in tables 6.13, 6.14, 6.15, 6.16 and 6.17. The results would be optimistic if we use the same software for training as well as testing. Thus, we have used 10 - cross validation to obtain the results. The performance of the binary and the non - binary models is evaluated using measures such as sensitivity, specificity, AUC and g-mean.

6.5.1 KC1 Result Analysis

Table 6.13 shows the results of validation of binary as well as non - binary models. The table shows that the binary models predicted using BN, NB and SVM give high g-mean values as compared to their equivalent non - binary models. Besides this, the binary model of SVM has also shown higher value of AUC (0.739) than its non - binary model (0.693). The performance of other binary models is comparable to their equivalent non - binary models. The ROC curve for the binary model predicted using SVM is shown in figure 6.3. Thus, we conclude that thresholds are quite effective and can be used by researchers and practitioners for alarming the classes that are at high-risk. In addition, SVM can be used for validating the threshold values when obtained for the metrics of some other proprietary software of similar nature.

Table 6.13: Validation Results for KC1

ML Technique	Validated Model	Sensitivity	Specificity	AUC	G-mean	Cut-off point
NB	Non-Binary	0.69	0.667	0.769	66.52	0.134
	Binary	0.724	0.69	0.713	69.66	0.479
BN	Non-Binary	0.707	0.621	0.736	64.9	0.694
	Binary	0.741	0.678	0.702	69.49	0.401
RF	Non-Binary	0.741	0.724	0.815	71.99	0.45
	Binary	0.69	0.678	0.716	67.13	0.591
SVM	Non-Binary	0.63	0.782	0.693	69.59	0.5
	Binary	0.81	0.667	0.739	72.08	0.5
MLP	Non-Binary	0.69	0.69	0.803	67.77	0.441
	Binary	0.655	0.667	0.723	64.94	0.592

6.5.2 Ivy Result Analysis

Table 6.14 shows the results of validation when the thresholds of the metrics of Ivy are validated using the same software. The table shows the performance of the binary and the non - binary models of Ivy. Among the models predicted using non - binary models, NB and BN have outperformed with the highest (same) values for sensitivity (0.75), specificity (0.744) and g-mean (51.17). Besides this, their values of AUC are also quite high (0.799 for NB and 0.791 for BN). In addition to NB and BN, MLP has also shown high values of sensitivity (0.725), specificity (0.721) and AUC (0.794). Among the binary models, we can see that RF and MLP have outperformed

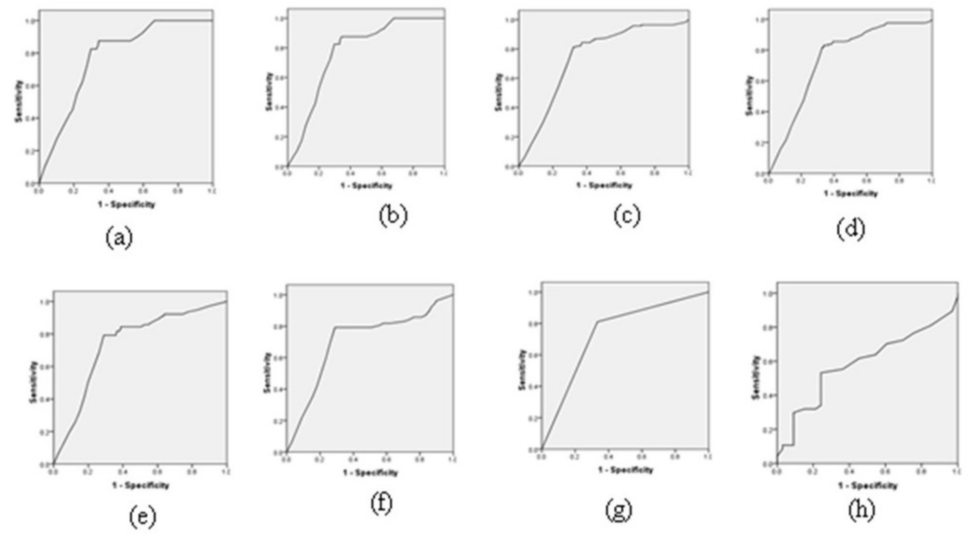


Figure 6.3: ROC Curves for Binary Models: (a) IVY: RF, (b) IVY: MLP, (c) ANT: NB, (d) ANT: RF, (e) TOMCAT: NB, (f) TOMCAT: RF, (g) KC1: SVM, (h) SAKURA: MLP

their corresponding non - binary models with higher values for specificity, AUC and g-mean, whereas same values for sensitivity. The values of specificity, AUC and g-mean for the binary model of RF are 0.724, 0.776 and 49.03 respectively whereas for the binary model of MLP, the values are 0.728, 0.773 and 49.23 respectively. Their ROC curves are shown in figure 6.3. Other binary models have shown comparative results when compared with their corresponding non - binary models. Hence, we concluded that our proposed methodology of using the threshold values stands strong for Ivy software as well, in addition to KC1.

Table 6.14: Validation Results for Ivy

ML Technique	Validated Model	Sensitivity	Specificity	AUC	G-mean	Cut-off point
NB	Non-Binary	0.75	0.744	0.799	51.17	0.005
	Binary	0.725	0.724	0.764	49.03	0.485

ML Technique	Validated Model	Sensitivity	Specificity	AUC	G-mean	Cut-off point
BN	Non-Binary	0.75	0.744	0.791	51.17	0.056
	Binary	0.725	0.724	0.763	49.03	0.505
RF	Non-Binary	0.725	0.683	0.735	46.46	0.05
	Binary	0.725	0.724	0.776	49.03	0.044
SVM	Non-Binary	-	-	0.5	-	-
	Binary	-	-	0.5	-	-
MLP	Non-Binary	0.725	0.721	0.794	48.81	0.086
	Binary	0.725	0.728	0.773	49.23	0.236

6.5.3 Inter - Project Validation

Tables 6.15, 6.16 and 6.17 show the results of inter - project validation. We conducted inter - project validation (threshold values of metrics obtained from one software is applied on the other software) on the projects of similar nature or having similar characteristics. This helps to prove the effectiveness and strength of the proposed methodology. Therefore, to conduct inter - project validation, we divided the five OSS (Ivy, Ant, Tomcat, JEdit and Sakura) to two different categories as discussed below. We selected these software as they are large enough to perform inter - project validation and also their source code is available in the open source repositories.

Category 1: The three software, namely Ivy, Ant and Tomcat are categorized together as they have certain features that are common amongst them. Thus, we have validated the threshold values of metrics of Ivy on Ant and Tomcat. In other words, Ant and Tomcat are converted into binary using threshold values of metrics of Ivy. Following are the features common amongst the three software:
All these are OSS, obtained from Promise data repository. They are released under

the Apache Software License. They are written in Java programming language, i.e. they have been developed in the same environment.

Tables 6.15 and 6.16 show the results of inter - project validation on Ant and Tomcat. Among the non - binary models of Ant, we observe that the best results are shown by MLP and BN with the highest values of sensitivity (0.753), specificity (0.753) and g-mean (65.26). The AUC for MLP is highest (0.812) followed by the AUC of BN (0.803). The binary models of NB and BN predicted have shown exactly the same performance. Among all the binary models, RF has shown the highest specificity (0.729), which is also higher than the specificity of its corresponding non - binary model. Similarly, specificity of binary NB model (0.708) is also higher than its non - binary model. Thus, overall the binary models of NB and RF have outperformed their corresponding non - binary models. Other binary models have shown comparable performance with their non - binary models. Their ROC curves are shown in figure 6.3.

Among non - binary models of Tomcat, all the models have shown good results (except SVM), the best being of BN and MLP. The model predicted using BN has shown the highest sensitivity (0.753) along with the high values of AUC (0.777) and g-mean (44.63). The model predicted using MLP has shown highest specificity (0.74), AUC (0.801) and g-mean (46.02). Among the binary models, NB and RF have shown better performance than their non - binary models. The binary model of NB has shown the higher values of specificity (0.738) and g-mean (44.36), whereas the binary model of RF has shown the higher values of sensitivity (0.792), g-mean (45.5) and the same value of specificity (0.711). The ROC curves of NB and RF are shown in figure 6.3. Overall, we conclude that the binary models predicted using RF, followed by NB (for all the three software; Ivy, Ant and Tomcat) have outperformed their non - binary models. Thus, validating the threshold values of Ivy on Ant, Tomcat and obtaining significantly good results give us stronger confirmation and proof of the effectiveness of the proposed methodology.

Table 6.15: Validation Results for Ant

ML Technique	Validated Model	Sensitivity	Specificity	AUC	G-mean	Cut-off point
NB	Non-Binary	0.759	0.699	0.788	61.82	0.012
	Binary	0.717	0.708	0.742	60.65	0.629
BN	Non-Binary	0.753	0.753	0.803	65.26	0.113
	Binary	0.717	0.708	0.745	60.65	0.631
RF	Non-Binary	0.719	0.758	0.786	64.38	0.25
	Binary	0.729	0.699	0.745	60.75	0.407
SVM	Non-Binary	-	-	0.636	-	-
	Binary	-	-	0.717	-	-
MLP	Non-Binary	0.753	0.753	0.812	65.26	0.194
	Binary	0.645	0.729	0.743	59.61	0.385

Table 6.16: Validation Results for Tomcat

ML Technique	Validated Model	Sensitivity	Specificity	AUC	G-mean	Cut-off point
NB	Non-Binary	0.779	0.659	0.772	42.2	0.003
	Binary	0.688	0.738	0.74	44.36	0.408
BN	Non-Binary	0.753	0.713	0.777	44.63	0.019
	Binary	0.714	0.708	0.718	43.19	0.309
RF	Non-Binary	0.714	0.711	0.736	43.42	0.05
	Binary	0.792	0.711	0.705	45.5	0.2
SVM	Non-Binary	-	-	0.5	-	-
	Binary	-	-	0.5	-	-
MLP	Non-Binary	0.74	0.74	0.801	46.02	0.065
	Binary	0.792	0.704	0.752	45.07	0.126

Category 2: The two software, namely JEdit and Sakura are categorized together as they have certain common characteristics. Thus, thresholds obtained using JEdit software have been validated on Sakura. Following are the features common amongst JEdit and Sakura:

The target domain of these software is same, i.e. they are text editors for programmers. The LOC for both these software is also similar (15227 for Sakura editor and 12535 for JEdit). Table 6.17 shows the results of inter - project validation on Sakura using the threshold values of JEdit. We observe that the performance of the non - binary models is well appreciating as some of the values of sensitivity and specificity are quite low. After validating the binary models, we obtained comparable results as are the results of non - binary models. Some of the models have also shown better sensitivity and specificity. Due to a number of advantages of using thresholds over the metric models (as discussed in chapter 2) for prediction of risky classes, we conclude that the threshold methodology should be used even if the validation results obtained using the threshold methodology are comparable to the validation results of metric models.

Table 6.17: Validation Results for Sakura

ML Technique	Validated Model	Sensitivity	Specificity	AUC	G-mean	Cut-off point
NB	Non-Binary	0.574	0.576	0.617	56.67	0.008
	Binary	0.617	0.606	0.62	60.24	0.331
BN	Non-Binary	0.617	0.364	0.459	48.17	0.554
	Binary	0.255	0.636	0.446	43.3	0.589
RF	Non-Binary	0.574	0.606	0.615	58.09	0.65
	Binary	0.574	0.545	0.581	55.2	0.485
SVM	Non-Binary	0.83	0.303	0.566	59.08	0.5
	Binary	0.617	0.515	0.566	55.94	0.5

ML Technique	Validated Model	Sensitivity	Specificity	AUC	G-mean	Cut-off point
MLP	Non-Binary	0.553	0.576	0.459	54.48	0.554
	Binary	0.617	0.545	0.586	57.4	0.482

6.6 Discussion

In this chapter, we have identified the thresholds of OO metrics for early detection of fault prone classes. We have used a methodology based on LR to calculate the threshold values for the metrics of a proprietary NASA dataset, KC1 and two open source PROMISE software, Ivy and JEdit. The thresholds are calculated at different risk levels between 0.01 to 0.15 and the performance results are compared to identify the most appropriate risk level. Using the threshold values obtained at the most appropriate risk level, binary models are constructed and validated using various ML techniques (BN, NB, RF, SVM and MLP). Besides this, inter - project validation is also carried out using three OSS, Apache Ant, Apache Tomcat and Sakura. The results of this work are summarized as follows:

1. Univariate LR is conducted to find the metrics significant in predicting fault prone classes. As threshold values are calculated for the metrics of KC1, Ivy and JEdit, univariate LR is applied on these software. We found CBO, RFC, WMC and LOC to be the significant predictors of fault proneness for both the software, KC1 and Ivy. In addition to these metrics, LCOM is also found to be significant for JEdit. These metrics can be used by researchers in their studies to bring out effective results.

2. For KC1, the best or the most appropriate risk level that gave threshold values of all the metrics within the observation range (i.e. positive values) is 0.15. This means for classes with metric values lower than corresponding threshold values, the probability of fault occurrence is lower than 15%. The threshold values of OO metrics at risk level 0.15 are 7.17 for CBO metric, 2.88 for RFC metric , 4.74 for WMC metric and 119.93 for LOC metric.

   For Ivy, the threshold values at the risk level 0.07 and above, are within the observation range of all the metrics. But we selected the lowest value, i.e. 0.07 as the appropriate risk level. This is because at risk level of 0.07, the risk of fault occurrence is lower than the probability of 7%. As the value of risk level increases, risk of fault occurrence also increases.

   For JEdit, we observed that the lowest risk level for which all the threshold values are within the range is 0.02. The threshold values at 0.02 risk level are 14.49, 15.96, 50.39, 174.18 and 245.18 for the metrics WMC, CBO, RFC, LCOM, and LOC respectively.

3. Inter - project validation done by validating the threshold values of Ivy on Ant and Tomcat, whereas threshold values of JEdit on Sakura concluded that the threshold methodology can be used on the software of similar characteristics.

Chapter 7

Identifying Threshold Values of Object Oriented Metrics for Change Prediction

7.1 Introduction

There are numerous advantages of using the threshold methodology to predict the risky classes as discussed in chapter 2. Chapter 2 also discusses the advantages of thresholds over the metric models to predict the classes that are more prone to changes and faults. Despite the advantages of thresholds, the researchers have obtained the threshold values of OO metrics on quality attributes such as fault proneness [29, 39, 69, 121, 129, 131], but no study till date has obtained the thresholds for OO metrics to predict change proneness. Studies need be conducted to identify the potential usage of threshold values for predicting change proneness. Hence, in this chapter, the thresholds of OO metrics of two OSS, Freemind and Xerces are identified which will be used for predicting the classes that are prone to changes. The methodology used in this chapter is same as used in chapter 6 to calculate the threshold values of OO metrics for change prediction. We have also identified the change prone classes using the metric based models constructed with various ML techniques so that a comparison can be drawn between the two methodologies. This chapter is divided into three main parts:

1. Assessment of the accuracy of the threshold methodology: We have validated the models based on thresholds using various ML techniques such as AB, LB, MLP, NB, RF and CRT. We have also validated the traditional metric based models using the same ML techniques and compared the results. This comparison allows to assess the effectiveness of the proposed threshold methodology.

2. Validation of the threshold values on different releases of the same software: The thresholds obtained from Freemind 0.9.0 and Xerces 2.9.0 are validated on the different releases of Freemind and Xerces software so that we can assess the applicability of the threshold values on various releases of the software.

3. Validation of threshold values on different software: The thresholds obtained from Freemind 0.9.0 and Xerces 2.9.0 are also validated on the similar nature software, Frinika 0.2.0 and Xalan 2.6.0 respectively so that we can externally validate the applicability of the identified threshold values. This allows to obtain generalized and well-formed results.

The results indicate that there exist thresholds for OO metrics and these thresholds can be applied by software engineers on similar nature software systems to find the change prone design and code areas that may require corrective action.

This chapter is organized as follows: Section 7.2 explains the research background, emphasizing on the variables used, empirical data collection along with the descriptive statistics. Following this, we present the research methodology in section 7.3. Section 7.4 explains the univariate and the threshold results. The results of validation are explained in section 7.5. The comparison between binary and non - binary models is conducted in section 7.6. The results of this chapter are summarized in section 7.7. Finally, the the results of studies in literature are compared with the results of the studies in this work in section 7.8.

The results of this chapter are published in [102].

7.2 Research Background

In this section, we have summarized the independent and dependent variables used in this chapter. We have also explained the software used along with their descriptive statistics. The statistical technique used and the methodology followed for calculating the threshold values have also been explained.

7.2.1 Dependent and Independent Variables

We have used a suite of OO metrics including the popularly used CK metrics [36] as the independent variables. For these metrics, threshold values are calculated. Using these threshold values, the classes which are more prone to changes are identified. Thus, the dependent variable used in this study is change proneness.

7.2.2 Empirical Data Collection

This work analyses four OSS: Freemind, Frinika, Xerces and Xalan, developed using Java programming language. We investigated various releases of each software: 4 releases each of Freemind and Xerces, 2 release of Frinika and Xalan, making a total of 12 releases. We briefly describe description each software: a) Freemind is a mind mapper and hierarchical editor. It serves various purposes such as it keeps the track of the projects, acts as workplace for Internet research and can be used for essay writing and brainstorming; b) Frinika is a complete music workstation which provides the end user with a complete platform for creating music with their computers; c) Apache Xerces is intended for creating and maintaining XML parsers and other related software components; d) Xalan is an XSLT processor which is intended at transformation of XML documents into HTML format, text document, or other XML file types. We studied the widely used releases of the aforementioned software. The summary of characteristics of each software (including the total number of classes and the number of classes changed) is provided in chapter 2. The diagrammatic representation of the distribution of change in each software is shown in figure 7.1. We calculated change between the successive releases of each software. The process of change collection is explained in detail in chapter 2.

Use of Datasets

We carried out three analyses using the datasets as explained below (see figure 7.2):

1. We computed the threshold values of OO metrics on Freemind 0.9.0 and Xerces 2.9.0.

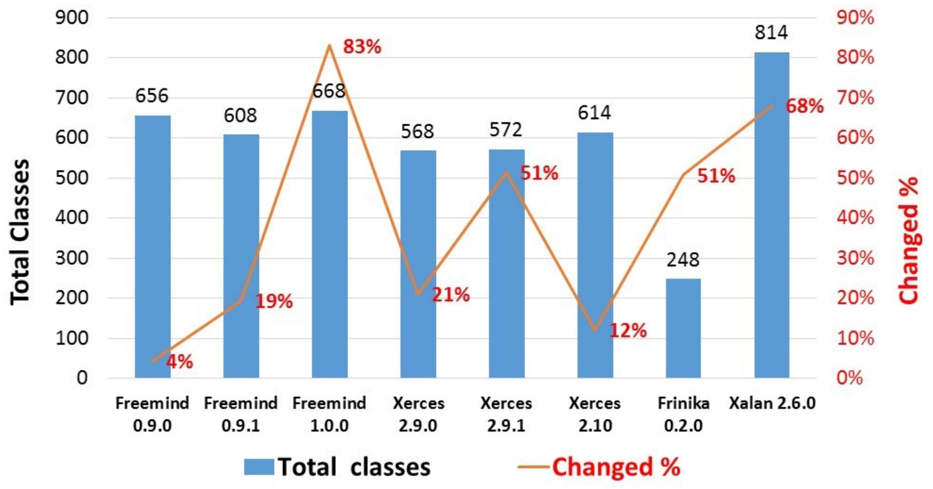


Figure 7.1: Representation of Change in Various Software

2. We validated the threshold values of the OO metrics on successive releases of Freemind (0.9.1, 1.0.0) and Xerces (2.9.1, 2.10.0). It is observed in literature that training the model from its current release (data) and validating its immediate next release gives better accuracy [130, 154]. However, in this study, we used the initial release to predict the change prone classes in the future releases. For example, we calculated the threshold values for the metrics of initial release of Freemind 0.9.0 and Xerces 2.9.0 and performed the validation on the subsequent or future releases of the respective software (i.e. Freemind 0.9.1, 1.0.0 and Xerces 2.9.1, 2.10.0). We did not train the predecessor release to validate its next release because we did not obtain the positive values of thresholds for Freemind 0.9.1, 1.0.0 and Xerces 2.9.1 and 2.10.0.

3. We validated the threshold values of the OO metrics of Freemind 0.9.0 and Xerces 2.9.0 on similar nature software, Frinika 0.2.0 and Xalan 2.6.0 respec-

tively. The external validation of Freemind is done on Frinika as they possess some common characteristics such as both are open source, free, written in Java and licensed under GNU GPL. Similarly there are common characteristics between Xerces and Xalan such as both are open source, written in Java and licensed to the Apache Software Foundation. Also their basic functionality is similar, revolving around XML documents.

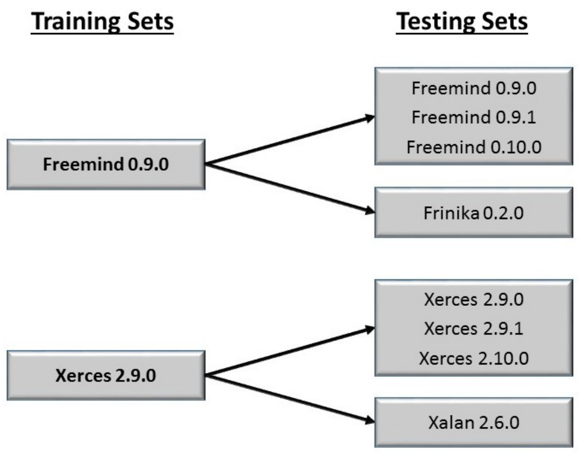


Figure 7.2: Demonstration of Training and Testing Sets

7.2.3 Descriptive Statistics

Tables 7.1, 7.2, 7.3 and 7.4 show the descriptive statistics (minimum (Min.), maximum (Max.), mean (Mean), median (Median) and standard deviation (SD)) which provide significant insights and allow to infer important conclusions about all the software used. Tables 7.1 and 7.3 show the descriptive statistics of all the four software used; Freemind, Frinika, Xerces and Xalan. Whereas, tables 7.2 and 7.4 show the descriptive statistics of various releases of Freemind and Xerces. Following important observations can be drawn from the tables:

- The size of the software is measured in terms of LOC. The number of source code lines for different releases of Freemind ranges from 3- 1513 (Freemind 0.9.0), 3 - 1525 (Freemind 0.9.1) and 3-1499 (Freemind 1.0.0). We can observe that maximum value of LOC for all the releases of Freemind is approximately the same. For Xerces, the range of LOC is 0-7714 (Xerces 2.9.0), 0-7814 (Xerces 2.9.1) and 0-9080 (Xerces 2.10.0). This shows that Xerces is a medium sized software.

- The median of DIT for all the releases of Freemind and Frinika is more than 0, i.e. 2. This indicates that at least more than half of the classes have a parent class. Also, we observe that mean of DIT for Freemind 0.9.0, 0.9.1 and 1.0.0 is 2.39, 2.4 and 2.4 respectively, whereas, for Frinika, it is 3.32. This shows that among the classes that have DIT more than 0, many of the classes have DIT more than 2 (for Freemind) or 3 (for Frinika). Thus, inheritance is widely used in these software. Similar results were shown by other studies [63, 153]. Zhou et al. [153] examined Eclipse 2.0 and concluded that out of a total of 5,225 classes, 3,340 have a parent class. Similarly, Gyimothy et al. [63] examined 3,192 classes in Mozilla 1.6 and found that more than half of the classes were subclasses.On the other hand, the median is 0 for all the releases of Xerces and Xalan. Besides this, the mean is 0.6 for all releases of Xerces and 0.79 for Xalan. These values of median and mean indicate that at least half of the classes are not using inheritance. On the remaining half of the classes, we cannot conclude anything. Overall, we can say that Freemind and Frinika have used inheritance to a much larger extent as compared to Xerces and Xalan.

- The value of CBO which measures the interaction between the classes is high for all the software used except Frinika (for example, it is 213 for all the releases of Freemind, between 85 to 95 for all the releases of Xerces and 169 for Xalan. This indicates that there is high interaction between classes.
- The LCOM metric has shown high values for all the releases of Xerces (upto 7297) and Xalan (upto 6045). Whereas, for all the releases of Freemind and Frinika, the value of LCOM is upto 100. This shows that the classes in Xerces and Xalan are more cohesive as compared to the classes in other two software.

Table 7.1: Descriptive Statistics of Freemind 0.9.0 and Frinika 0.2.0

	Freemind 0.9.0					Frinika 0.2.0				
	Mean	Median	SD	Min.	Max.	Mean	Median	SD	Min.	Max.
CBO	11.82	8	14.7	0	213	7.62	2	61.8	0	32
NOC	0.35	0	1.5	0	18	0.53	0	16.3	0	12
NOM	0.5	0	4.3	0	63	0.97	0	17.6	0	20
NOA	0.99	0	2.9	0	37	2.54	1	25.5	0	28
NIM	7.7	4	12.9	0	129	13.84	3	110	0	79
NIV	2.51	1	6	0	105	9.25	2	75.1	0	135
NLM	8.2	4	13.6	0	129	14.81	3	117.5	0	79
RFC	115.3	31	229.3	12	918	23.94	5	189	0	120
NLDM	0.33	0	1.2	0	21	0.79	0	17	0	9
NPRM	1.04	0	2.3	0	16	0.75	0	16.9	0	15
NPROM	0.53	0	1.9	0	23	0.36	0	16	0	10
NPM	6.29	3	11.6	0	116	12.9	3	102.8	0	73
LOC	87.71	38	156.2	3	1513	167.15	40	1319.5	3	1538
DIT	2.39	2	1.6	1	7	3.32	2	30.4	1	4
LCOM	36.55	33	36.4	0	100	93.64	50	735.3	0	100
WMC	15.47	7	28.8	0	250	28.25	6	222.9	0	163

Table 7.2: Descriptive Statistics of Various Releases of Freemind

	Freemind 0.9.1					Freemind 1.0.0				
	Mean	Median	SD	Min.	Max.	Mean	Median	SD	Min.	Max.
CBO	11.9	8	14.9	0	213	11.9	8	14.8	0	212
NOC	0.4	0	1.5	0	18	0.4	0	1.6	0	18
NOM	0.5	0	4.6	0	64	0.6	0	5.4	0	80
NOA	1	0	2.9	0	37	1.1	0	3.2	0	46
NIM	7.5	4	12	0	129	7.6	4	12.8	0	130
NIV	2.5	1	6.1	0	105	2.5	1	6.1	0	105
NLM	8	4	12.8	0	129	8.1	4	13.8	0	130
RFC	118.9	33	232.7	12	918	110.2	32.5	222.2	12	918
NLDM	0.3	0	1.3	0	21	0.3	0	1.2	0	20
NPRM	1	0	2.3	0	17	1	0	2.2	0	17
NPROM	0.6	0	1.9	0	23	0.6	0	1.9	0	23
NPM	6	3	10.6	0	116	6.3	3	11.9	0	117
LOC	87	37.5	156.4	3	1525	88	38.5	160.6	3	1499
DIT	2.4	2	1.6	1	7	2.4	2	1.5	1	7
LCOM	37.2	33	36.4	0	100	36.3	33	36.5	0	100
WMC	15.4	7	28.4	0	250	15.3	7	29	0	254

Table 7.3: Descriptive Statistics of Xerces 2.9.0 and Xalan 2.6.0

	Xerces 2.9.0					Xalan 2.6.0				
	Mean	Median	SD	Min.	Max.	Mean	Median	SD	Min.	Max.
CBO	10.4	5	13	0	88	9.53	6	15.2	0	169
NOC	0.5	0	3	0	52	0.51	0	2.2	0	29
NOM	0.9	0	3.5	0	49	0.99	0	3.9	0	67
NOA	3.6	1	11.8	0	161	11.44	0	42.4	0	334
NIM	11.8	7	14.4	0	123	9.56	4	16	0	131
NIV	3.9	1	8.5	0	78	2.59	1	4.7	0	41
NLM	12.7	8	15	0	125	10.56	5	16.2	0	132
RFC	29.2	15	39.2	0	296	25.98	16.5	33.6	0	345
NLDM	1.1	0	3.8	0	40	0.84	0	2.9	0	35
NPRM	1	0	3.1	0	29	0.64	0	1.7	0	19

	Xerces 2.9.0					Xalan 2.6.0				
	Mean	Median	SD	Min.	Max.	Mean	Median	SD	Min.	Max.
NPROM	1.3	0	3.6	0	43	0.56	0	2.9	0	41
NPM	9.3	5	10.9	0	77	9	4	14.4	0	120
LOC	413.3	93.5	878.5	0	7774	437.38	128	786	0	4330
DIT	0.6	0	0.7	0	5	0.79	0	1	0	5
LCOM	139.6	13	509.5	0	7297	126.78	6.5	543.3	0	6045
WMC	12.9	8	15.4	0	126	11.36	6	16.1	0	122

Table 7.4: Descriptive Statistics of Various Releases of Xerces

	Xerces 2.9.1					Xerces 2.10.0				
	Mean	Median	Std. Dev.	Min.	Max.	Mean	Median	Std. Dev.	Min.	Max.
CBO	10.2	5	12.9	0	88	10	5	13.2	0	93
NOC	0.5	0	2.9	0	52	0.5	0	2.9	0	52
NOM	1	0	3.6	0	49	0.8	0	3.4	0	49
NOA	3.6	1	11.7	0	161	3.3	1	11.6	0	161
NIM	11.7	7	14.5	0	123	11.1	6	15.4	0	127
NIV	3.9	1	8.4	0	77	4.2	1	8.8	0	83
NLM	12.7	8	15	0	125	12.8	8	15.6	0	129
RFC	29.1	15	39	0	301	28.8	15	41.9	0	432
NLDM	1.1	0	3.8	0	40	1.1	0	3.7	0	40
NPRM	1	0	3.1	0	29	1.1	0	4.2	0	78
NPROM	1.3	0	3.6	0	44	1.3	0	3.6	0	46
NPM	9.2	5	10.9	0	77	9.5	6	11.1	0	77
LOC	415.4	96.5	889.4	0	7814	402.9	90	888.4	0	9080
DIT	0.6	0	0.7	0	5	0.6	0	0.7	0	5
LCOM	138.3	12.5	509.8	0	7297	147.3	11	576.3	0	7781
WMC	12.8	8	15.4	0	126	13.1	8	16	0	130

7.3 Research Methodology

In this section, we focus on the data analysis techniques used for validating the models and the methodology used for calculating the thresholds of metrics.

7.3.1 Data Analysis Techniques Used

In this study, we have used one statistical and six ML techniques to validate various models. The statistical technique used in LR (the detailed explanation of LR is given in chapter 2. We have used univariate LR to find the metrics that are significantly associated with change proneness. Besides this, univariate LR is also used to find the threshold values of various OO metrics. The ML techniques used are briefly described in this section. Default setting of WEKA tool (`http://www.cs.waikato.ac.nz/ml/weka/`) is used for this purpose.

- LB and AB: Boosting is a technique of converting a weak learning classifier to a strong learning classifier [56]. In other words, we combine a number of weak hypotheses to get a better classification performance. For this, equal weights are assigned to all the training examples and then the weights of the incorrectly classified examples are increased on each round so that a weak learner is forced to focus on the hard examples in the training set. This is the main concept behind one of the algorithms of boosting, AB. One of the variants of AB is LB. The cost functional of LR is applied on AB algorithm to obtain LB algorithm [123].

- NB: It is a widely used inductive learning technique for ML and data mining

[150]. It is also called as a probabilistic classifier as it uses Bayes theorem for the purpose of classification. Bayes theorem finds the probability of an event occurring given the probability of another event that has already occurred. NB is the simplest BN where the classification variable is the parent node [43]. All the other variables act as child node of this parent node. Thus, unlike BN which can have many parent and child nodes, NB network has only one parent node and several child nodes as demonstrated in figure 7.3. No other connections are permissible in a NB network. We can observe from the figure that all the child

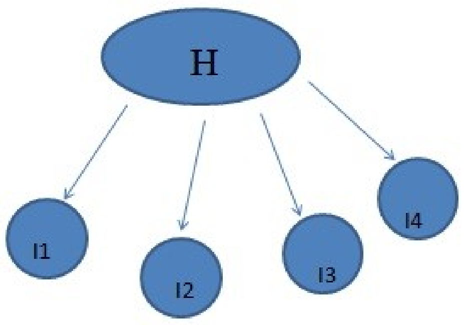


Figure 7.3: Naive Bayes Network

nodes are independent from each other. Thus, NB is based on the assumption of independence between the independent variables. Due to this assumption, it has two important advantages [33]: (1) it is easy to be constructed since the structure is known apriori and (2) it leads to efficient classification process.

- RF: It is a combination of a number of decision trees or classification trees. We can mention the number of decision trees we want in the forest. Each tree in the forest gives its own decision, known as the vote for that tree and the majority wins [21]. There are various advantages of using random forest such as high accuracy, works on large datasets, very little processing is required etc.

- MLP: It is a feedforward ANN trained with the error back propagation algorithm. It is an advancement to the perceptron neural network model with one or two hidden layers. There are two passes in the error back propagation algorithm: a forward pass and a backward pass. During the forward pass, the weights of the network are all fixed. During the backward pass, the weights are all updated and adjusted according to the error computed. The process is repeated until the performance is acceptable [123].

7.3.2 Calculation of Threshold Values

The Bender method known as VARL [12] which is based on LR is used for the computation of threshold values. This same methodology is also used in chapter 6. Thus, the concept, explanation and the description of VARL is provided in section 6.3.1.

7.4 Analysis of Univariate Logistic Regression and Threshold Methodology

In this section, we briefly discuss the results of univariate LR and threshold methodology. Univariate analysis of OO metrics is the initial step carried out to obtain a set of significant metrics for which the thresholds are calculated. Then, using the thresholds, we have converted the data into binary and applied CFS on the binary data to obtain the best subset of metrics. For validating the threshold values, we constructed various ML models for the binary datasets. We evaluated and compared the results of validation for the binary models with the results of the non - binary models. Non - binary models are obtained when we predict the change prone classes using the traditional metric based model building process. This basic approach followed in this study is also explained diagrammatically in figure 7.4.

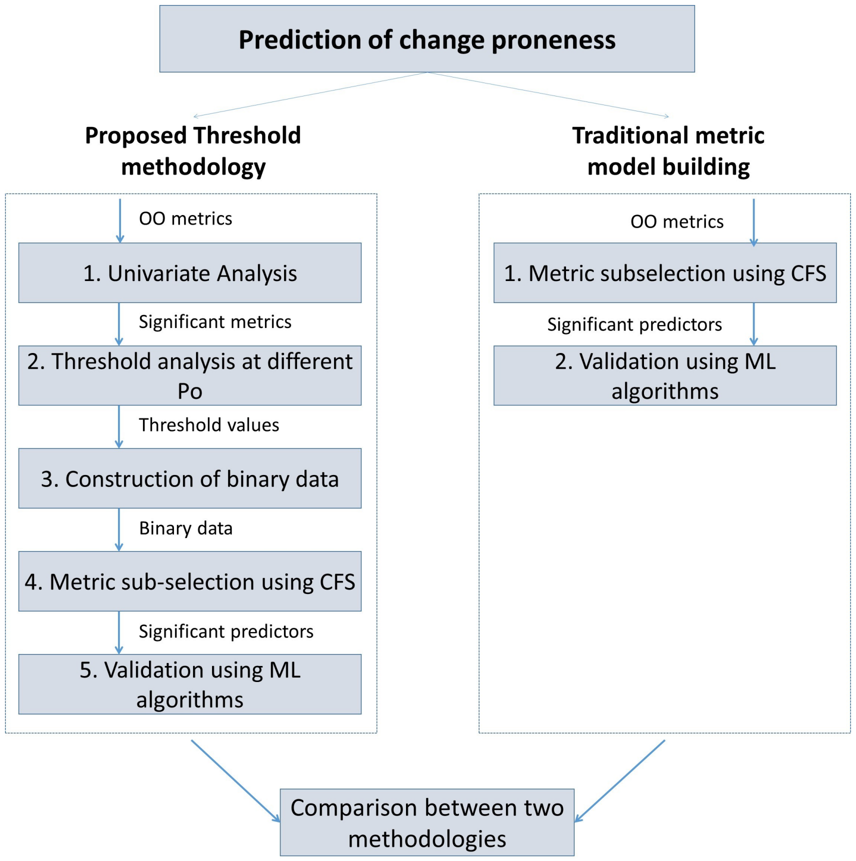

Prediction of change proneness
Proposed Threshold methodology
Traditional metric model building
OO metrics
1. Univariate Analysis
Significant metrics
2. Threshold analysis at different Po
Threshold values
3. Construction of binary data
Binary data
4. Metric sub-selection using CFS
Significant predictors
5. Validation using ML algorithms
OO metrics
1. Metric subselection using CFS
Significant predictors
2. Validation using ML algorithms
Comparison between two methodologies

Figure 7.4: Basic Approach Followed

7.4.1 Univariate Analysis

We conducted univariate analysis to identify the metrics which are significant in predicting change proneness. For the significant metrics, the threshold values are calculated. Since, the threshold values are calculated for the metrics of Freemind 0.9.0 and Xerces 2.9.0, univariate analysis is conducted on these software (table 7.5). For each metric, the values of coefficient (β), constant (α), and statistical significance (Sig.) are given. In this chapter, we are determining the significant predictors of change proneness at the 95% confidence level (i.e. Sig. value < 0.05 for the metric to be significant). The same confidence level is taken by number of other studies in literature [63, 130, 129]. Table 7.5 shows the significant metrics of both Freemind and Xerces in bold.

Table 7.5: Univariate Results

	Freemind 0.9.0			Xerces 2.9.0		
Metrics	α	β	Sig.	α	β	Sig.
CBO	-3.570	0.033	**0.000**	-1.879	0.047	**0.000**
NOC	-3.048	-0.095	0.620	-1.323	0.010	0.754
NOM	-3.159	0.061	**0.002**	-1.350	0.032	0.203
NOA	-3.305	0.135	**0.001**	-1.363	0.012	0.124
NIM	-3.352	0.026	**0.001**	-1.769	0.031	**0.000**
NIV	-3.141	0.022	0.213	-1.983	0.046	**0.000**
NLM	-3.457	0.031	**0.000**	-1.879	0.030	**0.000**
RFC	-3.374	0.002	**0.003**	-2.054	0.022	**0.000**
NLDM	-3.260	0.316	**0.003**	-1.885	0.068	**0.005**
NPRM	-3.463	0.214	**0.000**	-1.792	0.143	**0.000**
NPROM	-3.282	0.214	**0.000**	-1.91	0.102	**0.000**
NPM	-3.332	0.029	**0.001**	-1.742	0.022	**0.008**
LOC	-3.475	0.003	**0.000**	-1.770	0.001	**0.000**
DIT	-3.674	0.225	0.056	-1.739	0.515	**0.000**
LCOM	-3.925	0.018	**0.001**	-1.758	0.003	**0.001**
WMC	-3.486	0.016	**0.000**	-1.846	0.036	**0.000**

7.4.2 Threshold Analysis

For the significant metrics of Freemind 0.9.0 and Xerces 2.9.0, the threshold values are calculated at different risk levels (Po values), i.e. 0.01, 0.05, 0.08 and 0.1 (tables 7.6 and 7.7 respectively). We can see from tables 7.6 and 7.7 that VARL varies largely as the value of Po changes. For a small increase in the value of Po, VARL increases to a large extent. This shows that Po plays a significant role in calculating threshold values. Table 7.6 shows that the threshold values at Po = 0.01 lay outside the observation range of all metrics (negative values); therefore, we cannot use the thresholds at this level. Similarly, for Xerces, the threshold values at 0.01, 0.05 and 0.08 are outside the observation range. Thus, the potential threshold values for Freemind are at Po = 0.05, 0.08 and 0.1. For Xerces, the threshold values at Po = 0.1 are considered.

Table 7.6: Threshold Values of Freemind 0.9.0

Metrics	VARL at 0.01	VARL at 0.05	VARL at 0.08	VARL at 0.1
CBO	-31.064	18.956	34.171	41.599
NOM	-23.543	3.517	11.748	15.767
NOA	-9.556	2.671	6.39	8.206
NIM	-47.812	15.675	34.987	44.414
NLM	-36.714	16.534	32.731	40.638
RFC	-610.56	214.781	465.826	588.388
NLDM	-4.225	0.999	2.588	3.363
NPRM	-5.29	2.423	4.769	5.915
NPROM	-6.136	1.577	3.924	5.069
NPM	-43.556	13.364	30.678	39.13
LOC	-373.373	176.854	344.218	425.925
LCOM	-37.229	54.476	82.37	95.988
WMC	-69.32	33.848	65.228	80.548

Table 7.7: Threshold Values of Xerces 2.9.0

Metrics	VARL at 0.01	VARL at 0.05	VARL at 0.08	VARL at 0.1
CBO	-57.79	-22.669	-11.986	3.072
NOM	-91.165	-37.917	-21.721	1.11
NOA	-56.785	-20.901	-9.986	5.4
NIM	-90.637	-35.615	-18.878	4.713
NLM	-115.505	-40.474	-17.652	14.518
RFC	-39.855	-15.58	-8.196	2.212
NLDM	-19.602	-8.059	-4.548	0.401
NPRM	-26.325	-10.142	-5.219	1.72
NPROM	-129.687	-54.656	-31.834	0.336
NPM	-2825.12	-1174.44	-672.347	35.399
LOC	-5.546	-2.341	-1.366	0.009
LCOM	-945.707	-395.48	-228.116	7.8
WMC	-76.364	-30.512	-16.565	3.094

7.5 Validation Result Analysis

To assess the effectiveness of threshold values, we constructed the models using various ML techniques and evaluated their performance using various measures such as AUC, sensitivity (Sens.), specificity (Spec.) and g-mean (GM). In other words, we converted the data into binary using the threshold values at different Po values and then constructed ML models for this binary data. If the values of a particular metric are above its corresponding threshold values, then the metric values are changed to 1 (referring that it can be more change prone), else 0. For example, if threshold of CBO = 3.072 (table 7.7), then the values above 3.072 are converted to 1 and the values less than 3.072 are converted to 0. After converting the data to binary, CFS is applied to obtain the best subset of metrics. As discussed in section 7.4.2, the potential threshold values for Freemind are obtained at Po = 0.05, 0.08 and 0.1. Thus, three sets of binary data are obtained at each Po value and CFS is applied on each binary set. For Xerces 2.9.0, CFS is applied on only a single binary data obtained at Po=0.1. Table 7.8 lists the metrics selected by CFS for all the binary datasets of Freemind 0.9.0 and Xerces 2.9.0.

Table 7.8: Metrics selected by CFS

Software	Metrics selected
Freemind 0.9.0	
At 0.05	CBO, NLM, NPROM, WMC
At 0.08	CBO, NOA, NLM, NLDM, NPRM, NPROM, WMC
At 0.1	NOM, NOA, NLDM, NPRM, NPROM, WMC
Xerces 2.9.0	
At 0.1	NIV, NPROM, CBO, RFC, NPM
Inter-release validation	
Freemind 0.9.1	CBO, NPM
Freemind 1.0.0	NLDM, NPROM, NPM, LCOM, WMC
Xerces 2.9.1	CBO, RFC, LOC
Xerces 2.10.0	NLDM, WMC, CBO, RFC, NPM, LOC
External Validation	
Frinika 0.2.0	CBO, NLDM, LOC, LCOM
Xalan 2.6.0	NPRM, NPROM, RFC, NPM, LOC

For Freemind, to identify the most appropriate Po value, we validate the threshold values at each of these Po values on Freemind 0.9.0. In other words, we constructed various ML models to predict the change prone classes of the same release as on which the thresholds are determined (table 7.9). The Po at which the ML models show the best performance is selected as the most appropriate Po value (table 7.10). Then, the threshold values at that Po value are validated on different releases of Freemind (table 7.11).

Similarly for Xerces, the only potential threshold values are obtained at Po = 0.1 (discussed in section 7.4.2). Thus, we used the thresholds at 0.1 to predict the change prone classes of the same release (Xerces 2.9.0) and different releases of Xerces (table 7.12).

In this section, we discuss the results of validation of both Freemind and Xerces. The validation methodology is also shown diagrammatically in figure 7.5.

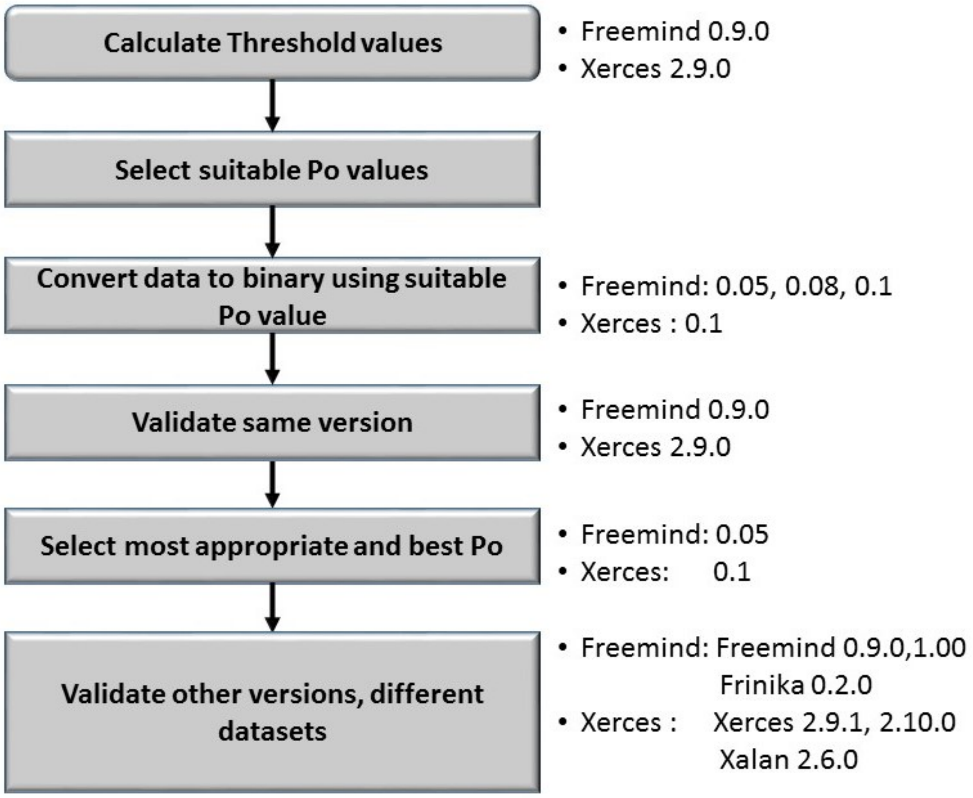

Calculate Threshold values
• Freemind 0.9.0
• Xerces 2.9.0
Select suitable Po values
Convert data to binary using suitable Po value
• Freemind: 0.05, 0.08, 0.1
• Xerces : 0.1
Validate same version
• Freemind 0.9.0
• Xerces 2.9.0
Select most appropriate and best Po
• Freemind: 0.05
• Xerces: 0.1
Validate other versions, different datasets
• Freemind: Freemind 0.9.0,1.00 Frinika 0.2.0
• Xerces : Xerces 2.9.1, 2.10.0 Xalan 2.6.0

Figure 7.5: Threshold Validation Methodology

7.5.1 Results of Validation of Freemind

Table 7.9 shows the results of validation at different Po values (0.05, 0.08 and 0.1). To select the most suitable Po value, we compared the AUC, sensitivity, specificity and g-mean values across the three Po values for all the ML techniques. Table 7.9 shows that CRT has given consistent result for all the Po values. Thus, we have compared the performance of the other five ML techniques at different Po values. Table 7.10 demonstrates the results of comparison. In table 7.10, a value 1 in each (Po - performance measure) cell represents that for a given technique, the result is best at that Po value for the performance measure, vis-a-vis the result at other Po values for the same performance measure. For example, AB, LB and MLP have shown higher values of AUC at 0.05 as compared to 0.08 or 0.1, therefore the value of cell for performance measure AUC at Po = 0.05 in table 7.10 is 3. Similarly, the values of sensitivity for all the ML techniques are higher at 0.05 than at 0.08 or 0.1, therefore the value of cell for sensitivity at 0.05 is 5. Overall, the best results are obtained at Po = 0.05 for 3 out of 4 performance measures (AUC, sensitivity and g-mean). The other Po values have the best results for 1 performance measure each. Thus, we concluded that the threshold values at 0.05 are most suitable for predicting change prone classes. We used these threshold values to predict the change prone classes of different releases of Freemind as well change prone classes of a different software. At 0.05, MLP has shown the best results with the highest values of AUC, sensitivity and g-mean.

Table 7.9: Validation Results of Various Threshold levels on Freemind

Technique	Freemind 0.9 (Po=0.05)				Freemind 0.9 (Po=0.08)				Freemind 0.9 (Po=0.10)			
	AUC	Sens. %	Spec. %	GM %	AUC	Sens. %	Spec. %	GM %	AUC	Sens. %	Spec. %	GM %
AB	0.68	72.4	79.7	75.9	0.655	62.1	79.9	69.9	0.669	58.6	81.7	68.2
LB	0.67	72.4	79.4	75.8	0.619	55.2	86.8	67.5	0.624	55.2	91.1	68.7
MLP	0.716	72.4	79.9	76	0.672	72.4	61.4	66.5	0.645	69	54.5	60.9
NB	0.686	72.4	78.6	75.4	0.716	69	85.8	76.5	0.695	65.5	81.3	72.6
RF	0.632	65.5	83.4	73.4	0.636	58.6	89.6	70.9	0.651	55.2	91.2	68.8
CRT	0.68	69	80.9	74.4	0.68	69	80.9	74.4	0.68	69	80.9	74.4

Table 7.10: Selection of Best Po for Freemind

Po / Performance Measure	AUC	Sens.	Spec.	GM
0.05	3	5	1	3
0.08	2	1	1	3
0.1	1	0	4	0

Inter - Release Validation of Freemind

Table 7.11 shows the results of validating the threshold values of metrics of Freemind 0.9.0 on the different releases, Freemind 0.9.1 and 1.0.0. Since, the threshold values at 0.05 are found to be most suitable, the metrics of Freemind 0.9.1 and 1.0.0 are converted to binary using these threshold values. CFS is again applied on the binary datasets (of different releases) and the metrics selected to predict change proneness are listed in table 7.8. For Freemind 0.9.1, CRT has shown the highest AUC, specificity and g-mean. MLP follows CRT by giving the highest value for sensitivity, whereas second highest values for all the other measures. Similarly, for Freemind1.0.0, CRT outperformed other models with the highest values of all the measures. MLP has shown the second best performance. Thus, we concluded that for validating the threshold values when applied on different releases of Freemind or similar software, CRT and MLP may be used. Researchers and practitioners may apply the thresholds obtained from a particular release on the successive releases to alarm the classes that fall outside the acceptable risk level.

Table 7.11: Validation Results of Potential Thresholds on Freemind

Software	Freemind 0.9.1				Freemind 1.0.0			
Technique	AUC	Sens. %	Spec. %	GM %	AUC	Sens. %	Spec. %	GM %
AB	0.661	60.2	64.7	62.3	0.643	70.1	54	61
LB	0.659	60.2	64.9	62.4	0.658	62.9	69	65.8
MLP	0.716	65.3	65.7	76	0.685	63	70.9	66.2
NB	0.665	62.7	65.3	64	0.662	62.9	69	65.8
RF	0.678	61	65.1	63.3	0.67	59.3	70.8	64.5
CRT	0.69	63.6	80	70.8	0.718	63.94	80.79	72.4

7.5.2 Results of Validation of Xerces

For Xerces, only the potential threshold values are found at Po = 0.1. Thus, we validated these threshold values on the same release, i.e. Xerces 2.9.0 and the other releases, i.e. Xerces 2.9.1 and 2.10.0. In other words, we converted the metrics of Xerces 2.9.1 and 2.10.0 to binary using the threshold values and CFS is applied to the binary datasets (table 6.8). Then different ML models are constructed to validate the threshold values (table 7.12). For Xerces 2.9.0, MLP and CRT outperformed the other models. MLP has shown the highest AUC and specificity, whereas CRT has shown the highest sensitivity and g-mean. For Xerces 2.9.1, MLP outperformed the other models with the highest value of AUC, sensitivity and g-mean. Its specificity value is also the second highest (76.6%). For Xerces 2.10.0, MLP has again outperformed with the highest values of AUC, specificity and g-mean. Other model have shown comparable performance for all the binary datasets. We concluded that MLP and CRT can be used by researchers to validate the threshold values. When validating different releases, we obtained comparable or even better results of performance measures. For example, Xerces 2.10.0 has shown higher AUC and sensitivity for all the ML techniques than the values obtained for Xerces 2.9.0. Xerces 2.9.1 has shown higher specificity values of all the classifiers as compared to the specificity values of Xerces 2.9.0. The better values of performance measures on different releases indicate that the thresholds can be effectively used on different releases of the same software.

Table 7.12: Validation Results of Potential Thresholds on Xerces

Technique	Xerces 2.9.0				Xerces 2.9.1				Xerces 2.10.0			
	AUC	Sens. %	Spec. %	GM %	AUC	Sens. %	Spec. %	GM %	AUC	Sens. %	Spec. %	GM %
AB	0.758	77.5	62.9	69.5	0.713	61.6	77.3	68.6	0.818	91.9	66.1	76.9
LB	0.756	79.2	62.1	69.6	0.707	61.6	77.3	68.6	0.824	91.9	67.2	77.6
MLP	0.758	70.8	67.4	69.1	0.719	63.3	76.6	69.3	0.836	87.8	70.9	78.5
NB	0.754	79.2	61.4	69.2	0.712	61.2	77.7	68.5	0.825	91.9	67.6	77.9
RF	0.751	75.8	66.7	71	0.71	70.7	66.9	68.8	0.823	91.9	67.4	77.8
CRT	0.749	79.2	62.94	70.13	0.707	61.6	77.3	68.6	0.776	91.9	66.7	77.3

7.5.3 External Validation

For evaluating the prediction accuracy of the thresholds, they are also validated on different software, Frinika and Xalan. The threshold values of the metrics of Freemind 0.9.0 are evaluated on Frinika 0.2.0 and the threshold values of the metrics of

Xerces 2.9.0 are evaluated on Xalan 0.6.0. This is termed as external validation. The results of the CFS on their binary datasets are given in table 7.8. Table 7.13 presents the results for external validation. We observed that the results of validating Frinika are almost similar for all the techniques. LB, MLP, NB and RF have shown exactly the same results for sensitivity (66.7%), specificity (63.1%) and g-mean (64.8). Overall, we can say that we obtained competitive and consistent results for Frinika. On validating the threshold values of Xerces on Xalan, MLP has shown highest AUC, sensitivity and g-mean. Other techniques including AB, NB and RF have shown good and comparable performance. Thus, MLP can be used for validating the thresholds on different software.

Table 7.13: External Validation

Software	Frinika 0.2.0				Xalan 2.6.0			
Technique	AUC	Sens. %	Spec. %	GM %	AUC	Sens. %	Spec. %	GM %
AB	0.637	64.3	63.1	63.7	0.701	60.1	74.2	66.4
LB	0.621	66.7	63.1	64.8	0.705	59.9	75.8	66.9
MLP	0.638	66.7	63.1	64.8	0.707	72.2	63.1	67.3
NB	0.628	66.7	63.1	64.8	0.697	61.9	72.3	66.7
RF	0.607	66.7	63.1	64.8	0.697	60.3	74.6	66.7
CRT	0.612	66.7	63.1	64.8	0.669	63.7	65.8	64.7

7.6 Comparison with non-binary models using statistical tests

Since the proposed methodology of identifying change prone classes using the thresholds of metrics is a relatively newer technique than the traditional metric model building, thus, it is very essential to judge the effectiveness and accuracy of the thresholds. For this, we constructed the ML models for predicting change proneness for non-binary data as well. The models are constructed using the metrics selected by CFS (table 7.14). The construction of non-binary models means the construction of traditional metric based models without any inclusion of the threshold values. Whereas, in section 7.5, we discussed the performance of various binary models (binary data is obtained by applying the threshold values of metrics). Hence, the intent is to evaluate whether the performance of binary models is comparable to non - binary models. The performance of non - binary models is evaluated using AUC. We compared the AUC of both binary as well as non-binary models.

Table 7.14: Metrics Selected by CFS (of Various Non - Binary Models)

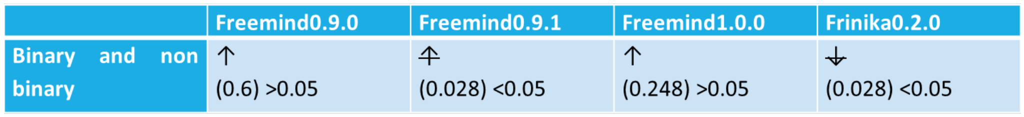
Software	Metrics selected
Freemind 0.9.0	RFC, NPRM, NPROM, NLDM, WMC
Xerces 2.9.0	NIV, NPRM, CBO, RFC, NPM, LOC
Frinika 0.2.0	NOA, NPRM, LCOM
Xalan 2.6.0	NPROM, RFC, LCOM, LOC, NOM, NOA

We checked if there is a significant difference in the performance of both the types of models using a statistical test known as Wilcoxon Signed Rank test. The detailed description of Wilcoxon Signed Rank test is given in chapter 2. If the binary models give comparable results with the non - binary models, we conclude that the threshold methodology is effective. The intent here is not to demonstrate that binary models outperform the non - binary models, but instead, we are only comparing the performance of both the types of models. If comparable performance is achieved, then researchers may use the thresholds of metrics for predicting change proneness as there are various advantages of using thresholds over traditional metric model building.

The results of comparison between the binary and non - binary models are shown in figures 7.6 and 7.7. The significance value for each of the comparison is shown in brackets. Figure 7.6 shows that the significance level obtained when the binary models of Freemind 0.9.0 and 1.0.0 are compared with their non - binary models is more than 0.05. Thus, null hypothesis is accepted, i.e. there is no significant difference in the performance of binary and non - binary models of Freemind 0.9.0 and 1.0.0. But, the significance level obtained when the binary models of Freemind 0.9.1 and Frinika 0.2.0 are compared with their non - binary models is less than 0.05, concluding that there is significant difference in the performance of binary and non - binary models. Now, based on the sum of the ranks (given by Wilcoxon Signed Rank test), the results show that the binary models of Freemind 0.9.1 are significantly better than the compared non - binary models whereas, the reverse is true in case of Frinika 0.2.0. Figure 7.7 shows that the significance level obtained when the binary models of all Xerces and Xalan datasets are compared with their non - binary models is more than 0.05. This concludes that there is no significant difference in the performance of binary as well as non - binary models.

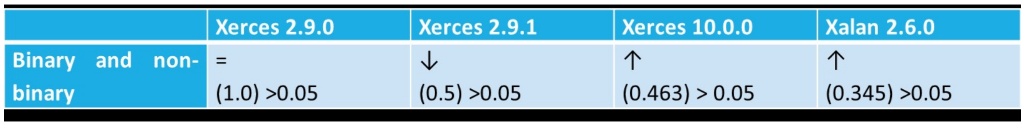
Overall, based on the results of comparison in figures 7.6 and 7.7, we concluded that there is no significant difference between the performance of binary and non-binary models.This shows the threshold methodology can be effectively utilized to identify change prone classes in the upcoming releases of the same software or other similar software.

$\uparrow$ binary models significantly better than the compared non binary models,$\uparrow$ binary models better than the compared non binary models, $\downarrow$ binary models significantly worse than the compared non binary models, ↓ binary models worse than the compared non binary models, = implies same (equal)performance



	Freemind0.9.0	Freemind0.9.1	Freemind1.0.0	Frinika0.2.0
Binary and non binary	↑ (0.6) >0.05	⍏ (0.028) <0.05	↑ (0.248) >0.05	⍖ (0.028) <0.05

Figure 7.6: Results of Wilcoxon Test on Freemind and Frinika



	Xerces 2.9.0	Xerces 2.9.1	Xerces 10.0.0	Xalan 2.6.0
Binary and non-binary	= (1.0) >0.05	↓ (0.5) >0.05	↑ (0.463) > 0.05	↑ (0.345) >0.05

Figure 7.7: Results of Wilcoxon Test on Xerces and Xalan

7.7 Discussion

In this chapter, we have studied and calculated the threshold values for various OO metrics. The classes whose OO metrics exceed the threshold values (alarming values) can be selected for focussed attention to improve the quality. A statistical methodology based on LR is used to calculate the threshold values of various OO metrics of two OSS, Freemind 0.9.0 and Xerces 2.9.0. The threshold values are validated using different ML techniques such as AB, LB, MLP, NB, RF and CRT. The important observations from this chapter can be summarized as:

1. The results of univariate LR showed that there is a significant relationship between the metrics and change proneness. Majority of the metrics are found to be significant in predicting change proneness.

2. There are effective threshold values for the OO metrics and there is significant effect of Po values on the threshold values. Threshold values change as we change the value of risk level (Po). Thus, risk level plays an important role in the calculation of threshold values and should be taken into consideration. For Freemind, Po= 0.05 is found to be the most appropriate, whereas for Xerces , Po= 0.1 is found to be appropriate.

3. The effectiveness of the proposed threshold methodology is judged on various releases of Freemind (Freemind 0.9.1, 1.0.0) and Xerces (2.9.1, 2.10.0). In addition to this, external validation on two different software, Frinika 0.2.0 and Xalan 2.6.0 is also carried out. Among a number of ML techniques used, MLP and CRT can be used for validating different releases of Freemind and Xerces as well as different software.

4. The performance of binary models is compared with the performance of non - binary models (models without thresholds) using a statistical test known as Wilcoxon signed rank test. The results of the test showed that there is no significant difference in the performance of binary and non - binary models. This shows that the threshold methodology can be effectively utilized on upcoming or the future releases of software.

7.8 Comparison with Studies in Literature

In this section, we compare the results of this work with the results of the previous studies. We have compared the results with respect to the following perspectives:

1. Univariate results: The comparison of the univariate analysis allows to identify the metrics which are significant in predicting change proneness.

2. Model Validation Results: The comparison of the model validation results allows to identify the techniques which outperform the other techniques and thus, can be used for the purpose of model construction for classifying the classes into change and non - change prone categories.

3. Threshold Results: This comparison allows to explore the threshold methodology used in other related studies.

7.8.1 Comparison of Univariate Results

Although there are a number of empirical studies that have constructed or proposed various models for predicting change prone classes, but very few studies have related OO metrics with change proneness. In other words, the univariate analysis which identifies the metrics which are significant predictors of change proneness is not conducted by many studies. Thus, the literature does not allow to conclude the significant predictors of change proneness. To the best of our knowledge, only the study by Zhou et al. [153] has related OO metrics with change proneness using univariate analysis. In this section, we compare the univariate results of this study [153] with the univariate results obtained in this work. The comparison will provide important insights about the relationship between independent and dependent variables. Table 7.15 shows the univariate results of the metrics of various software used in this work along with the univariate results of the study by Zhou et al. [153]. For the purpose of comparison, we can observe from the table that the univariate results of only the popularly used CK metrics [36] and LOC are presented. Other OO metrics are not used in every study of this work and also are not used by Zhou et al.[153]. Thus, inclusion of such metrics will not allow lead to fair comparison. Table 8.11 shows that for all the software, CBO, WMC and LOC are positively significant to change proneness. This implies that as the value of each of these metric increases, the extent of change proneness also increases. Similarly, RFC is also found to be significantly associated with change proneness in all the software except for all of the releases of Android. LCOM has also shown the similar result, i.e. it is found to be a significant predictor of change proneness in most of the software except for KC1 and Ivy. Similar results are also observed by the study in literature [153] , CBO, LCOM and RFC are found to be significant predictors of change proneness (WMC and LOC are not used in this study). The metric, NOC is mostly found to be insignificant in this work except for few releases of Android, whereas it is found to be significant in the study in literature [153]. We can observe from the table that the inverse relationship with change proneness is only shown by DIT. In other words, DIT is the only metric which is significant to change proneness but in an inverse manner. This implies that as the value of DIT increases, the extent of change proneness decreases. Overall, we can make the following two conclusions from this comparison:

1. All the metrics are either positively related to change proneness or are not related at all (insignificant), except for DIT which showed negative impact on change proneness.

2. CBO, WMC, LOC, RFC and LCOM are significant predictors of change proneness. These metrics can be used by researchers and practitioners while con-

structing various models for predicting change prone classes.

7.8.2 Comparison of Model Validation Results

There are very few studies in literature which have constructed prediction models using various ML techniques. We found that there are only two studies [58, 124] which have constructed ML models for predicting the classes which are more prone to changes. Thus, we compare the performance of the models in these studies with the performance of the ML models constructed in this work. For the purpose of evaluation, we have compared the performance using the performance measure, AUC as this measure is used by all the studies under the comparison. Besides this, AUC has a number of advantages over other performance measures such as it handles imbalanced and noisy data etc. which makes it an accurate performance measure. In this work, we found that the models predicted using RF, bagging and MLP (ANN) outperformed the other models predicted using different ML techniques. The primary reason for RF and bagging to outperform is the fact that they reduce the variance associated with the prediction. This is due to the reason that they are ensemble learners which combine multiple classifiers to improve the accuracy and the final outcome is the majority vote of the prediction results of all the classifiers. As the variance reduces, the over - fitting decreases which inturn leads the prediction error to decrease. Thus, overall it leads to improvement in the performance.

Table 7.15: Univariate Results Comparison

Metrics/ Software	Results of this work											Literature results: Zhou et al. [153]
	JTreeView	Android2.3	Android4.0	Android4.1	Android4.2	Android4.3	KC1	Ivy	JEdit	Freemind	Xerces	Eclipse
CBO	+	+	+	+	+	+	+	+	+	+	+	+
NOC	0	+	0	+	0	0	0	0	0	0	0	+
WMC	*	+	+	+	+	+	+	+	+	+	+	*
DIT	0	-	-	-	-	-	0	0	0	0	+	+
LCOM	+	+	+	+	+	+	0	0	+	+	+	+
RFC	+	0	0	0	0	0	+	+	+	+	+	+
LOC	+	+	+	+	+	+	+	+	+	+	+	*

‘+’ shows that the metric is significant to change proneness

‘-’ shows that the metric is significant to change proneness but in an inverse manner

’*’ shows that the metric is not used in the study

‘0’ shows that the metric is insignificant

Besides this, we noticed that the type of relationship between the independent and the dependent variables used in this study is unknown. In other words, it is highly non - linear and complex. For such type of data, RF and ANN are recommended as they are widely used to perform non - linear and statistical modeling [139]. In simple words, they have the implicit ability to detect non - linear relationships between the dependent and the independent variables. Thus, we found that RF, bagging and MLP have performed well on the datasets used in this study. In contrast, the studies in literature have shown the superior performance of the Bayesian models over the other ML models. Bayesian models which also include NB model have shown moderate performance on the datasets used in this work. Thus, we conclude that the researchers may use the prediction models constructed using RF, bagging and MLP (ANN) for classifying the classes into change and non - prone categories. These models can further be used to predict or classify the change prone of some other similar nature software as well.

7.8.3 Comparison of Threshold Results

To the best of the knowledge, we found that there is no study in literature which has explored the use of thresholds for predicting change prone classes. However, there is one study by Shatnawi [129] which has used thresholds to predict fault prone classes. Thus, we compare this study with one of our studies [100] where we have used thresholds to predict fault prone classes. Below, we discuss the main highlights of both the studies. They are also summarized in table 7.16.

1. Discussion of study by Shatnawi [129]:

    - Calculated the threshold values of CK metrics [36] of Eclipse software using the statistical methodology proposed by Bender [12].
    - Calculated the thresholds at different risk levels (Po): 0.05, 0.06, 0.065, 0.075, 0.10.
    - Found acceptable threshold values at risk level 0.06 and above. Assessed the effectiveness of these thresholds to identify the fault-prone classes using the CART decision trees.

2. Discussion of our study [100]:
    - Calculated the threshold values of CK [36] and LOC metrics of three software (KC1, Ivy, JEdit) using the same statistical methodology as is used in [129].
    - Calculated the thresholds at different risk levels (Po): 0.01, 0.015, 0.02, 0.05, 0.06, 0.08, 0.10, 0.015.
    - Found acceptable threshold values at different risk levels depending on the software. For KC1, acceptable risk level obtained is 0.15, for Ivy, it is 0.07 and for JEdit, it is 0.02.
    - Assessed the effectiveness of these thresholds to identify the fault-prone classes using various ML techniques: NB, BN, RF, SVM and MLP.

We have not compared the threshold values of metrics obtained in both the studies as the thresholds are highly dependent on software used (threshold values depend on the values of constant and coefficient of each metric which vary according to software).

In both the studies under comparison, different software are used and thus, the comparison of thresholds is not appreciated. In other words, we can only compare the threshold values of metrics of similar nature software.

Table 7.16: Comparison of Studies Exploring Threshold Methodology

Characteristics	Shatnawi [129]	Our Work[100]
Software	Eclipse	KC1, Ivy, JEdit
Metrics Used	CBO, RFC, WMC, DIT, NOC	CBO, RFC, WMC, DIT, NOC, LCOM, LOC
Significant Metrics	CBO, RFC, WMC	KC1: CBO, RFC, WMC,LOC
		Ivy: CBO, RFC, WMC,LOC
		JEdit: CBO, RFC, WMC, LOC, LCOM
Po values	0.05, 0.06, 0.065, 0.075, 0.1	0.01, 0.05, 0.06, 0.08, 0.1, 0.15
Threshold Methodology	Value of Acceptable Risk Level (VARL)	Value of Acceptable Risk Level (VARL)

Chapter 8

Identifying Threshold Values Using Receiver Operating Characteristics Curve

8.1 Introduction

The risk indicators or threshold values of various OO metrics can be identified which may be used for prediction of change and fault prone classes. There are two approaches identified for the threshold computation. One of the approaches is based on a statistical LR technique, whereas, the other approach makes the usage of ROC curves for the computation of the threshold values. The statistical LR approach has been discussed and used in chapters 6 and 7.

In this chapter, we have used the methodology based on ROC curves to compute the threshold values of various OO metrics. This technique has only been used by Shatnawi et al. [131] to identify the threshold values of OO metrics. Besides the work by Shatnawi et al. [131], this technique is not used in the field of Software Engineering and is only used in the field of Medical Science to make decisions. Besides this, to assess the effectiveness of both the approaches, we have also calculated the thresholds using the statistical approach and compared the results. We found that the ROC methodology is more effective than the methodology based on LR and can be used by researchers to calculate the threshold values.

The threshold values are calculated for the metrics of an open source, Linux based operating system, Android. The change data is collected from five official releases of Android (2.3, 4.0, 4.1, 4.2 and 4.3). The thresholds are calculated for the metrics of various releases and validated on their immediate successor releases. In other words, inter - release validation is carried out, i.e. the threshold models obtained from the data of a particular release are used to predict the change prone classes of the immediate successor release.

This chapter is organized as follows. Section 8.2 focuses on the independent and dependent variables used in this chapter. Section 8.3 describes in detail the empirical data collection method, along with the descriptive statistics of all the independent variables. Section 8.4 describes the methodology to calculate the threshold values. The results of the study are summarized in section 8.5, which also includes the results of inter - release validation. In section 8.6, the thresholds are found using the statistical approach and a comparison is drawn between the two methodologies. Finally, in section 8.7, we summarize the results of this chapter.

8.2 Independent and Dependent Variables

The threshold values are calculated for a suite of metrics consisting of 15 OO metrics including six CK metrics [36]. Thus, these are the independent variables in this work and the dependent variable is change proneness. Thus, using the threshold values of metrics, the change prone classes are predicted.

8.3 Research Methodology

In this section, we have explained the data used for empirical validation along with the descriptive statistics of various independent variables used.

8.3.1 Empirical Data Collection

In this study, we have used six widely used releases of an open source; Linux based operating system, Android. The releases used in this work are the same as were used in chapter 5, i.e. Android 2.3 (Gingerbread), 4.0 (Ice cream Sandwich), 4.1 (JellyBean), 4.2 (JellyBean), 4.3 (JellyBean) and 4.4 (KitKat). The criteria for selection of these releases are explained in chapter 5. The details of each release which include the total number of classes and the number of classes changed are given in chapter 2.

8.3.2 Descriptive Statistics

We calculated different statistics which include mean (Mean), median (Median), standard deviation (SD), skewness (Skew.), kurtosis (Kurtosis), minimum (Min.), maximum (Max.), and percentiles (25%ile and 75%ile) of all the metrics used (tables 8.1, 8.2, 8.3, 8.4 and 8.5). From the tables, we observed some important characteristics of different parameters such as size, inheritance, coupling, cohesion etc. as discussed below:

Table 8.1: Descriptive Statistics of Android 2.3

Metrics	Mean	Median	SD	Skew.	Kurtosis	Min.	Max.	25th %ile	75th %ile
CBO	6.5	4	9.9	4.89	38.36	0	150	1	8
NOC	0.96	0	7.8	17	325.36	0	178	0	0
NOM	1.54	0	7.23	13.59	261.4	0	188	0	1
NOA	4.63	0	16.13	9.04	112.38	0	302	0	3
NIM	10.1	4	23.02	8.29	94.97	0	339	2	10
NIV	3.62	1	8.89	7.94	87.06	0	130	0	4
NLM	11.63	5	24.5	7.62	81.86	0	347	2	11
RFC	59.42	13	127.34	3.13	9.91	0	850	4	35
NPRM	1.81	0	7.15	12.68	224.53	0	148	0	1
NPROM	0.65	0	2.83	13.9	277.29	0	71	0	0
NPM	8.35	4	16.98	6.57	61.12	0	252	2	8
LOC	169.16	48	414.83	8.35	111.23	1	9753	17	147
DIT	2.1	2	1.45	1.4	1.09	1	7	1	3
WMC	26.41	8	72.69	10.47	173.3	0	1983	3	24
LCOM	45.85	50	38.38	-0.04	-1.6	0	100	0	83

Table 8.2: Descriptive Statistics of Android 4.0

Metrics	Mean	Median	SD	Skew.	Kurtosis	Min.	Max.	25th %ile	75th %ile
CBO	6.91	4	10.48	5	42.61	0	167	1	8
NOC	0.98	0	9.01	20.76	513.67	0	289	0	0
NOM	1.4	0	6.58	13.83	278.6	0	190	0	1
NOA	4.73	0	17.46	9.65	120.12	0	302	0	3
NIM	10.44	4	23.56	9.8	151.74	0	498	2	10
NIV	3.91	1	9.26	7.61	82.88	0	149	0	4
NLM	11.84	5	25	9.05	131.9	0	509	2	12

Metrics	Mean	Median	SD	Skew.	Kurtosis	Min.	Max.	25th %ile	75th %ile
RFC	74.83	13	176.67	3.38	11.45	0	1182	4	38
NPRM	1.84	0	6.74	13.12	292.64	0	193	0	1
NPROM	0.65	0	3.01	17.23	434.61	0	94	0	0
NPM	8.46	4	17.25	7.8	103.13	0	358	1	8
LOC	185.38	47	484.77	8.2	107.52	1	11894	16	148
DIT	2.06	2	1.39	1.44	1.33	1	8	1	3
WMC	47.05	50	38.5	-0.08	-1.59	0	100	0	84
LCOM	27.2	8	77.41	12.57	246.48	0	2413	3	24

Table 8.3: Descriptive Statistics of Android 4.1

Metrics	Mean	Median	SD	Skew.	Kurtosis	Min.	Max.	25th %ile	75th %ile
CBO	7.31	4	11.8	5.36	44.1	0	178	2	8
NOC	1.04	0	9.75	21	525.88	0	321	0	0
NOM	1.36	0	6.23	14.19	304.11	0	190	0	1
NOA	4.88	0	17.82	9.49	117.16	0	302	0	3
NIM	10.89	4	26.12	10.71	177.8	0	581	2	10
NIV	4.23	1	10.56	8.14	93.36	0	168	0	4
NLM	12.25	5	27.64	10	157.1	0	592	2	12
RFC	82.86	14	192.87	3.27	10.54	0	1208	5	40
NPRM	2	0	7.12	10.4	172.51	0	160	0	1
NPROM	0.7	0	3.21	16.62	386.26	0	89	0	0
NPM	8.6	4	18.94	9.77	160.05	0	422	1	8
LOC	197.7	49	539.07	8.19	99.47	1	12358	18	150
DIT	2.07	2	1.39	1.41	1.28	1	8	1	3
WMC	47.26	53	38.1	-0.11	-1.58	0	100	0	83
LCOM	29.23	8	91.82	12.52	217.08	0	2517	3	24

Table 8.4: Descriptive Statistics of Android 4.2

Metrics	Mean	Median	SD	Skew.	Kurtosis	Min.	Max.	25th %ile	75th %ile
CBO	7.11	4	11.1	5.36	47.42	0	179	2	8
NOC	1.02	0	10.09	22.31	588.82	0	342	0	0
NOM	1.39	0	6.48	13.89	288.81	0	195	0	1
NOA	4.75	0	17.29	9.42	116.57	0	302	0	3
NIM	10.53	4	24.14	10.34	175.88	0	590	2	10
NIV	4.01	1	9.68	8.06	95.55	0	170	0	4
NLM	11.92	5	25.75	9.63	154.53	0	603	2	11
RFC	87.49	14	206.31	3.16	9.44	0	1224	4	40
NPRM	1.89	0	6.82	11.49	213.08	0	170	0	1
NPROM	0.64	0	2.86	17.12	437	0	91	0	0
NPM	8.48	4	17.87	8.88	138.98	0	424	1	8
LOC	187.84	47	497.04	8.11	103.23	1	11857	17	146
DIT	2.05	2	1.37	1.46	1.43	1	8	1	2
WMC	46.76	51	38.29	-0.08	-1.59	0	100	0	83
LCOM	27.41	8	79.65	12.91	251.75	0	2340	3	24

Table 8.5: Descriptive Statistics of Android 4.3

Metrics	Mean	Median	SD	Skew.	Kurtosis	Min.	Max.	25th %ile	75th %ile
CBO	7.12	4	10.66	4.88	40.96	0	189	2	8
NOC	1	0	10.1	23.19	637.13	0	353	0	0
NOM	1.37	0	6.75	14.33	293.54	0	195	0	1
NOA	4.4	0	17.07	10.27	137.03	0	305	0	2
NIM	10.29	4	22.37	10.41	197.55	0	601	2	10
NIV	3.98	1	8.83	7.2	79.72	0	185	0	4
NLM	11.66	5	23.97	9.43	165.3	0	615	2	11
RFC	85.62	14	203.85	3.24	10.14	0	1256	4	40
NPRM	1.8	0	5.67	7.4	80.09	0	107	0	1
NPROM	0.64	0	2.83	17.19	441.93	0	92	0	0
NPM	8.34	4	17.44	9.17	153.71	0	437	1	8

Metrics	Mean	Median	SD	Skew.	Kurtosis	Min.	Max.	25th %ile	75th %ile
LOC	180.53	46	474.75	8.31	113.64	1	12135	17	137
DIT	2.06	2	1.37	1.41	1.3	1	7	1	3
WMC	46.05	50	38.32	-0.06	-1.61	0	100	0	82
LCOM	26.66	8	76.01	13.64	290.76	0	2384	3	23

- Size: The size of each class measured in terms of LOC ranges from 1 to 9753 (Android 2.3), 1 to 11894 (Android 4.0), 1 to 12358 (Android4.1), 1 to 11857 (Android 4.2) and 1 to 12135 (Android 4.3). This shows that Android is a large - sized software.

- Inheritance: We can notice that the median of DIT for all the releases is greater than 0, showing that more than half of the classes have a parent class. Also, we observe that mean of DIT for all the releases is more than 2. This shows that among the classes that have DIT more than 0, many of the classes have DIT more than 2. On the contrary, it is observed that the median of NOC for all the releases is 0, showing that at least half of the classes do not have any child. This is also indicated by the percentile statistic, which shows that at least 75% of the classes have no children. Thus, seeing the statistics of DIT and NOC, it is difficult to comment on the extent of inheritance in the system.

- Cohesion: The LCOM measure (for all the releases), which counts the number of classes with no attribute usage in common, has high values (upto 100). Thus, the software supports high cohesion. Similar results were observed by other studies as well [5, 134].

- Coupling: The maximum values of coupling are 150, 167, 178, 179 and 189 for the releases 2.3, 4.0, 4.1, 4.2 and 4.3 respectively. This shows that there is high interaction between classes.

- Variations across classes: We can observe that for the metrics RFC, LOC and LCOM, there is significant difference between the lower 25th percentile, the median, and the 75th percentile, thus showing strong variations across classes. But for the majority of the metrics, the difference in the values for 25th percentile, the median, and the 75th percentile are not much. Thus, overall we can say that there are not many variations across classes.

8.4 ROC Analysis to Calculate Threshold Values

ROC curve is used for estimating and evaluating the performance of the predicted models. The ROC curve is defined as a plot of sensitivity on the y-coordinate versus its 1-specificity on the x- coordinate [44, 134].

In this study, we have used ROC curve to identify the threshold values of OO metrics. To plot the ROC curve, we need to define two variables: one binary (i.e., 0 or 1) and another continuous. Usually, the binary variable is the actual dependent variable (for e.g. change proneness) and the continuous variable is the predicted result of a test. When the results of a test fall into one of the two obvious categories, such as change prone or non - change prone, then the result is a binary variable (1 if the class is change prone, 0 if the class is non - change prone) and we have only one pair of sensitivity and specificity. But, in many situations, making a decision in binary is not possible and thus, we give the decision or result in probability (i.e. probability of correct prediction). Thus, the result is a continuous variable. In this scenario, we choose different cut - off points that make each predicted value (probability) as 0 or 1. In other words, we use different cut - off points to change the continuous variable into binary. If the predicted probability is more than the cut - off, then we make the probability as 1, otherwise 0. In other words, we say if the predicted probability is more than the cut - off, then the class is change prone, otherwise non - change prone. This procedure is carried for various cut - off points and values of sensitivity & specificity is noted at each cut - off point. Thus, using the (sensitivity, specificity) pairs, we construct the ROC curve. In other words, ROC curves display the relationship between sensitivity (true - positive rate) and 1-specificity (false - positive rate) across all possible cut - off values. We find an optimal cut - off point which is defined as the cut - off point where sensitivity equals specificity.

In this study, the binary variable is the dependent variable (change proneness) and the continuous variable is the metric used in the study. For each metric, we choose as many as different cut - off points and calculate the values of sensitivity and 1- specificity at each point. In other words, at each cut-off point, we have a pair of sensitivity and 1- specificity values. The ROC curve is constructed using these pairs by taking sensitivity on the y-coordinate and its 1-specificity on the x- coordinate. The optimal cut-off point (where sensitivity equals specificity) obtained from ROC curve is considered as the threshold value for the metric. The similar procedure is carried for all the metrics to obtain their threshold values. This same procedure for identifying the threshold values of metrics is also explained diagrammatically in figure 8.1.

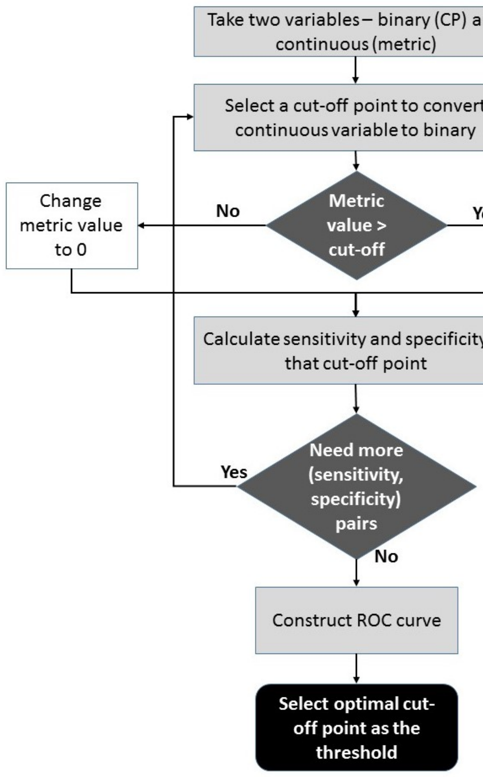

Take two variables – binary (CP) and continuous (metric)
Select a cut-off point to convert continuous variable to binary
Metric value > cut-off
No
Yes
Change metric value to 0
Change metric value to 1
Calculate sensitivity and specificity at that cut-off point
Need more (sensitivity, specificity) pairs
Yes
No
Construct ROC curve
Select optimal cut-off point as the threshold

Figure 8.1: Calculation of Threshold values

8.5 Results Analysis

In this section, we discuss the results which focus on the threshold analysis and the inter - release validation. The threshold values obtained using the methodology based on ROC are reported and discussed. This is followed by the results obtained when the thresholds for a particular release are validated on the immediate successor release.

8.5.1 Analysis and Interpretation of Threshold Values

We calculated the threshold values for the metrics of Android 2.3, 4.0, 4.1 and 4.2 using the methodology described in section 8.4. The threshold values are listed in table 8.6 for all the releases of Android. We cannot use these threshold values in practical, real life projects unless they are validated or their classification performance has been checked. We have measured the classification performance of using the threshold value to put the classes into change or non - change prone categories. For this, we first obtained the binary dataset corresponding to each release using the threshold values of the metrics of that release. For example, using the threshold values of Android 2.3, we obtained binary dataset of Android 2.3. Each metric is converted to binary by changing the its value to 1, if it is more than its threshold and 0, otherwise. To validate the threshold values, we constructed a model for each individual binary metric using a ML technique, RF. We measured the performance of each model using AUC which measures the classification performance of the threshold values to put classes into change and non - change categories. The AUC of random forest models for all the binary metrics of each release is listed in table 8.7. The AUC below or equal to 0.5 means no good classification [68]. In other words, the model having AUC less than or equal to 0.5 has no practical utility. We consider only those thresholds as the practical threshold values whose AUC is atleast more than or equal to 0.55. If AUC is less than 0.55, we conclude that the threshold values obtained for those metrics are not practical. From table 8.7, we can observe that for Android 2.3, only three RF models (corresponding to NOC, NOM and RFC) have given AUC less than 0.55. For Android 4.1, five models have shown AUC less than 0.55. Similarly, for both Android

4.0 and 4.2, four RF models have given AUC less than 0.55. Thus, the threshold values of the metrics corresponding to these models cannot be utilized to validate other projects. On the other hand, the threshold values of all the other metrics (having AUC of RF more than 0.55) have practical significance and can be used to validate other projects. In this chapter, these metrics are termed as 'acceptable metrics' and only they are used to validate different releases of the software.

Table 8.6: Threshold Values

Metric	Android 2.3	Android 4.0	Android 4.1	Android 4.2
CBO	3.5	4.5	4.5	4.5
NOC	0.5	0.5	0.5	0.5
NOM	0.5	0.5	0.5	0.5
NOA	0.5	0.5	0.5	0.5
NIM	4.5	4.5	4.5	4.5
NIV	1.5	1.5	1.5	1.5
NLM	5.5	5.5	5.5	5.5
RFC	13.5	14.5	17.5	15.5
NPRM	0.5	0.5	0.5	0.5
NPROM	0.5	0.5	0.5	0.5
NPM	3.5	3.5	5.5	3.5
LOC	52.5	56.5	65.5	60.5
DIT	1.5	1.5	1.5	1.5
LCOM	53.5	58.5	58.5	60.5
WMC	9.5	10.5	10.5	10.5

Table 8.7: AUC of Random Forest Model for Each Metric

Metric	Android 2.3	Android 4.0	Android 4.1	Android 4.2
CBO	0.6	0.603	0.598	0.601
NOC	0.512	0.506	0.532	0.51
NOM	0.542	0.508	0.555	0.568
NOA	0.577	0.557	0.588	0.563

Metric	Android 2.3	Android 4.0	Android 4.1	Android 4.2
NIM	0.564	0.564	0.518	0.568
NIV	0.57	0.579	0.552	0.568
NLM	0.578	0.57	0.563	0.554
RFC	0.543	0.554	0.566	0.566
NPRM	0.577	0.565	0.566	0.55
NPROM	0.563	0.496	0.501	0.482
NPM	0.606	0.556	0.515	0.527
LOC	0.615	0.579	0.605	0.584
DIT	0.592	0.508	0.504	0.516
LCOM	0.56	0.572	0.56	0.578
WMC	0.57	0.599	0.591	0.6

8.5.2 Analysis of Inter - Release Validation

We predicted the change prone classes in a particular release of the software by using the threshold values of the metrics of its immediate predecessor release. In other words, we evaluated the accuracy of the thresholds by validating them on the immediate successor release. For example, for predicting change prone classes for Android 4.1, we used the threshold values of its immediate predecessor, i.e. Android 4.0. We did not use the threshold values of a particular release to predict the change prone classes of all the subsequent releases. This is done as it is observed in literature that rather than using the metric prediction models constructed from the data in the first release and using them to predict change prone classes in the subsequent releases, the accuracy is higher if the prediction models are constructed / trained from the data of immediate predecessor release. Thus, we used the threshold values of: i) Android 2.3 to predict change prone classes in Android 4.0, ii) Android 4.0 to predict change prone classes in Android 4.1 and, iii) Android 4.1 to predict change prone classes in Android 4.2. In other words, using the threshold values of Android 2.3, 4.0 and 4.1, we converted the metrics of Android 4.0, 4.1 and 4.2 respectively into binary. In addition to validating the immediate successor release, we also validated the same release as well. For example, we used the threshold values of Android 2.3 to predict the change prone classes of Android 2.3 itself. We constructed RF model for each binary dataset and evaluated the performance using AUC. The results are shown in table 8.8. From table 8.8, we can observe the AUC obtained after validating the same release and the immediate successor release is comparable. This shows that since, the threshold values of the predecessor release can always be obtained, they can be utilized to predict change prone classes in the immediate next release.

Table 8.8: AUC of Random Forest Models

Use Threshold values of	Use Thresholds values on	AUC
Android 2.3	Android 2.3	0.701
	Android 4.0	0.684
Android 4.0	Android 4.0	0.672
	Android 4.1	0.702
Android 4.1	Android 4.1	0.705
	Android 4.2	0.692
Android 4.2	Android 4.2	0.682

8.6 Calculation of Thresholds Using Statistical Method

In this section, we have provided a comparison with the thresholds calculated using the statistical method proposed by Bender [12] known as VARL. The detailed description of this methodology is given in chapter 6. In this chapter, we have found the threshold values of Android 2.3, 4.0, 4.1 and 4.2 using the statistical method and compare the values with those obtained using the ROC methodology. We briefly explain the steps followed to calculate the threshold values for the metrics of Android 2.3. Similar steps can be taken to calculate the threshold values for the metrics of Android 4.0, 4.1 and 4.2.

Step 1: The first step involves the calculation of α (constant) and β (estimated coefficient) for each individual metric. For this, we applied univariate LR on Android 2.3. Table 8.9 shows the univariate results of Android 2.3.

Table 8.9: Univariate Results of Android 2.3

Independent Variable	β	α
CBO	0.066	-0.852
NOC	0.013	-0.446
NOM	0.018	-0.462
NOA	0.015	-0.503
NIM	0.021	-0.63
NIV	0.056	-0.624
NLM	0.02	-0.65
RFC	0	-0.453
NPRM	0.061	-0.535
NPROM	0.073	-0.48
NPM	0.02	-0.601
LOC	0.001	-0.608
DIT	-0.209	-0.007
WMC	0.008	-0.628
LCOM	0.011	-0.943

Step 2: After obtaining the values of α and β, we applied the VARL formula for calculating the threshold values. As discussed in chapter 6, Po is the acceptable risk level which takes the values 0.05, 0.01 etc. In this chapter, we have calculated the threshold values at different Po values, 0.01, 0.05, 0.08 and 0.1. The threshold values

of all the metrics are shown in table 8.10.

Table 8.10: Thresholds at Various Po Values

Metrics	VARL at 0.01	VARL at 0.05	VARL at 0.08	VARL at 0.1
CBO	-56.71	-31.7	-24.1	-20.38
NOC	-319.16	-192.19	-153.57	-134.71
NOM	-229.62	-137.91	-110.02	-96.4
NOA	-272.81	-162.76	-129.29	-112.95
NIM	-188.82	-110.21	-86.3	-74.63
NIV	-70.91	-41.44	-32.47	-28.09
NLM	-197.26	-114.72	-89.62	-77.36
RFC	- -	- -	- -	- -
NPRM	-66.56	-39.5	-31.27	-27.25
NPROM	-56.37	-33.76	-26.88	-23.52
NPM	-199.71	-117.17	-92.07	-79.81
LOC	-3987.12	-2336.44	-1834.35	-1589.22
DIT	21.95	14.05	11.65	10.48
LCOM	-332.01	-181.95	-136.3	-114.02
WMC	-496.54	-290.2	-227.44	-196.8

We can observe from table 8.10 that the threshold values lay outside the observation range (negative) of most of the metrics at all the risk levels. Thus, we cannot identify thresholds at any level. We obtained similar results for Android 4.0, 4.1 and 4.2. For all the releases of Android, the negative threshold values for all the metrics is obtained. Thus, they cannot be considered and put to any practical use.

8.7 Discussion

We calculated the threshold values of various OO metrics using the methodology based on ROC curves. These threshold values are calculated for the metrics of a

mobile operating system, Android. Threshold values are also calculated using the methodology based on LR as explained in chapter 6. This allowed to compare and assess the effectiveness of both the methodologies. We observed that the ROC methodology is effective in producing the threshold values within the range (positive values) of most of the metrics. These threshold values are validated on the same release (as on which the thresholds are obtained) and on the immediate successor release. The validation results obtained are comparable which concludes that the thresholds obtained using ROC methodology can be used to predict change prone classes of some other (successor) release of the same or similar software.
On the other hand, when we used the LR methodology to obtain the threshold values, we obtained negative threshold values of all the metrics. Thus, we found that statistical approach of calculating thresholds can produce out of range threshold values (negative) for some software , whereas, within the range (positive) threshold values for some others. In contrast, the ROC methodology used to calculate the threshold values can never produce the threshold values that are outside the observation range. In other words, the threshold values are always positive and fall within the minimum and maximum values of a particular metric. Thus, we conclude that ROC methodology is a more effective approach and should be used by researchers and practitioners in their study to calculate the threshold values of various metrics.

Chapter 9

Cross Project Change Prediction Using Open Source Projects

9.1 Introduction

There are several empirical studies in literature [8, 28, 53, 63, 66, 67, 79, 118, 124, 134, 152, 153] which have developed models for predicting change and fault prone classes. But these studies have constructed the model by training it using the historical data of the same project. The predicted model is used to further identify change and fault prone classes in the upcoming or future releases of the same or similar nature projects. However, prediction of the models using the training data from the same project is not always feasible as it might be possible that the adequate training data is not available or is not well collected [78, 141, 145, 155]. The possibility of non-availability of training data has led to the idea of performing cross - project predictions. There have been few studies [78, 114, 140, 141, 155] that have carried out cross - project prediction for predicting fault prone classes. But to the best of the knowledge, the work of cross project predictions for change proneness has not been done yet. Cross - project change prediction is defined as predicting the changes by training the models using the historical data of some other projects [145, 155]. In other words, to predict changes in a project B, we train the models using the data of the other project A, and then test them on the project B. We can observe that training and testing are performed on the different projects instead of using the same project. In this work, under cross - project predictions, we have considered two categories: inter - release predictions and inter - project predictions. Inter - release predictions refer to the predictions carried on two different releases of the same project. Whereas, inter - project predictions refer to the predictions carried on two different projects. In this work, we tried to investigate the following two issues or RQs:

- RQ1: How does the accuracy of the model differ when we train the model using the different releases of the same project (i.e. performing inter - release validation) or when we use different project (i.e. performing inter - project)?
- RQ2: Are the characteristics of datasets valuable for selecting the suitable training data for cross - project change prediction?

Aiming at the above questions, we conducted an experiment on 12 public datasets obtained from three projects. All the three projects are Apache projects (Apache POI, Apache Rave and Apache Abdera) and we considered four latest releases for each project, thus making a total of 12 datasets. We considered all the possible combinations (A, B) to perform cross - project predictions for change proneness. We employed a ML technique, LB to construct the prediction models (to measure the performance

of each cross - project predication). To determine the effectiveness of distributional characteristics in selecting the training data, we constructed a CRT and generated various classification rules. //The different sections of this chapter are: Following this section, we focus on the research background which focus on the variables and various performance measures used. In the next section, we explain the datasets along with the data collection process. Section 9.4 describes the basic four steps used in this chapter to conduct cross - project predictions.We discuss the results in section 9.5. Finally, this work is concluded in section 9.6.
The results of this chapter are published in [98].

9.2 Research Background

In this section, we state the independent and dependent variables used in this study along with the various performance measures that are used to evaluate the performance of the models.

9.2.1 Variables and Performance Measures Used

We have used the famous CK metrics [36] proposed as the independent variables. In addition to the metrics proposed by CK, size metric, SLOC is also used. The dependent variable is change proneness.
We have constructed a number of prediction models (for each possible train/test (A, B) combination) using a ML technique, LB. We evaluated each of these models using two performance measures, precision and AUC. Based on certain values of these performance measures, we have categorized each prediction as ‘successful/valid’ or

'not valid'.

9.3 Empirical Data Collection

We have used four releases each of three open source Apache projects, Abdera, POI and Rave. Each release is considered as a different dataset, thus making a total of 12 datasets. Apache Abdera (http://abdera.apache.org/) is an implementation of the Atom Syndication Format and Atom Publishing Protocol which are used for creating, editing, and publishing various web sources. Apache POI (http://poi.apache.org/) is used for making and modifying different file formats based upon OOXML and Microsoft's OLE 2. We can read/ write Word, Excel and PowerPoint files using Java. It is used in text extraction applications such web spiders etc. Apache Rave (http://rave.apache.org/) provides an extensible platform for using, integrating and hosting various OpenSocial and W3C Widget related features, technologies and services. Rave is popularly used as engine for Internet and Intranet portals. The releases which we have considered in this work are as follows (1) Abdera 1.0, Abdera 1.1, Abdera 1.1.1 and Abdera 1.1.2, (2) POI 3.0, POI 3.6, POI 3.7 and POI 3.9, (3) Rave 0.19, Rave 0.20.1, Rave 0.21.1 and Rave.22. The details of each of these releases which include the total number of classes and the number of classes changed in each release are given in chapter 2.

We briefly describe the change collection process as follows:

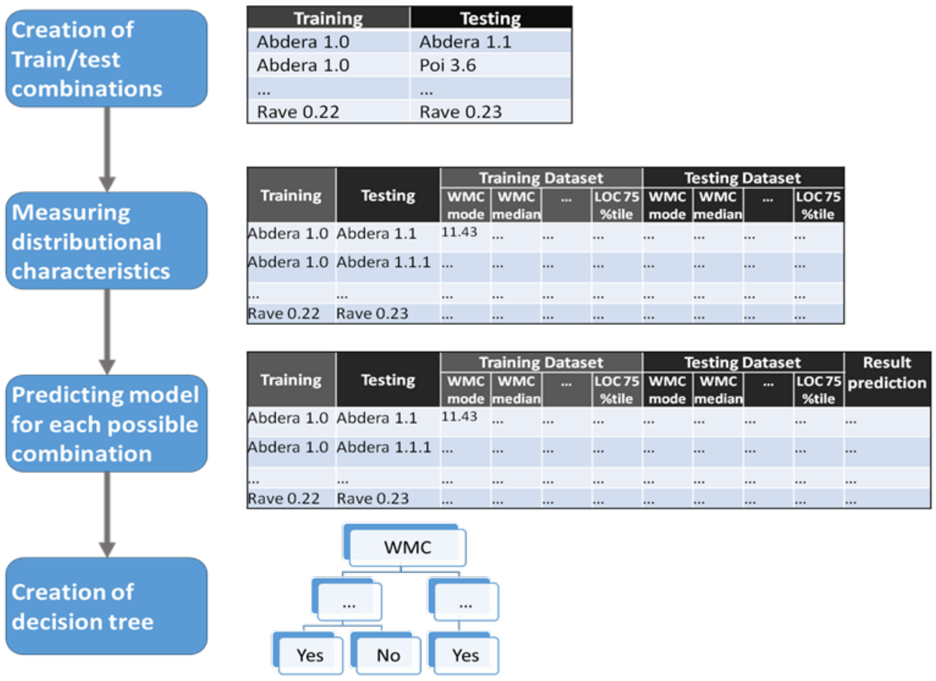
For collection of the data, we used a tool named Defect Collection and Reporting System (DCRS) [106], developed in Java by the students of Delhi Technological University, Delhi, India. The tool is used to generate the change reports for OSS hosted at Git repository. The brief functioning of the tool is explained below: //Two consecutive releases of software are taken and their source code is analyzed. Using Git commands, the source code is processed and change - logs are retrieved. A Change Log contains all the information concerned with the modifications that have been incurred from time to time in the source code. The modification in the source code could be due to various reasons such as defect-fixing, function/feature enhancements etc. Each modification is stored as an individual change record in the change log. An individual change record at the Git repository consists of: a) Change timestamp, b) Unique change identifier, c) Description of the change including the reason why the change has occurred, d) Source code files that are modified, along with the LOC changes for each modified file. Finally, these change records are mapped to source code classes. Only Java source code files are considered and other files such as xml and other resources are ignored. The change- report generated consists of the following fields:

- Name of the Java source file (Class name).
- A binary variable to indicate whether a class has been modified, i.e. ‘yes’ else ‘no’.

9.4 Research Methodology

In this section, we explain the basic four steps involved in this work. The first step is the creation of train/test combination or generation of datapoints. This is followed by measuring distributional characteristics for each train/test combination. The third step is the construction or prediction of ML model to determine the accuracy of cross - project predictions. Finally, a CRT is created to judge the suitability of characteristics of data sets for selecting the training data for cross - project change prediction. All the steps are explained diagrammatically in figure 9.1.

Figure 9.1: Basic Research Methodology


Creation of Train/test combinations
Measuring distributional characteristics
Predicting model for each possible combination
Creation of decision tree
WMC
...
...
Yes
No
Yes

Training	Testing
Abdera 1.0	Abdera 1.1
Abdera 1.0	Poi 3.6
...	...
Rave 0.22	Rave 0.23

Training	Testing	Training Dataset				Testing Dataset			
		WMC mode	WMC median	...	LOC 75 %tile	WMC mode	WMC median	...	LOC 75 %tile
Abdera 1.0	Abdera 1.1	11.43	...	...	...	...	...	...	...
Abdera 1.0	Abdera 1.1.1	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...	...
Rave 0.22	Rave 0.23	...	...	...	...	...	...	...	...

Training	Testing	Training Dataset				Testing Dataset				Result prediction
		WMC mode	WMC median	...	LOC 75 %tile	WMC mode	WMC median	...	LOC 75 %tile	
Abdera 1.0	Abdera 1.1	11.43	...	...	...	...	...	...	...	...
Abdera 1.0	Abdera 1.1.1	...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...	...	...
Rave 0.22	Rave 0.23	...	...	...	...	...	...	...	...	...

9.4.1 Creation of Train/Test Combinations

We have used three Apache projects (Apache POI, Apache rave, Apache Abdera), with four releases of each. A project with four releases is considered as four separate datasets and therefore, three projects will comprise of 12 datasets. To perform cross - project validation, one dataset among the 12 datasets is taken as the training set and some other dataset is taken as the testing set. We have considered the possible combinations of training and testing sets (A, B) if: 1) A and B belong to different projects or 2) A and B belong to same project, then B should be the later release than A [155]. In other words, we want the testing set to be the later release than the training release as it is illogical to perform testing on the former release and training on the later release. For example, we used Abdera 1.1.1 to predict the change prone classes of the later release, Abdera 1.1.2 and not the former release, Abdera 1.1. Therefore, the number of possible combinations with A,B,C…..N projects, having number of releases as a,b,c….n respectively is:

$${}^{T}P_2 - \sum_{i=A}^{N} \frac{V_i * (V_i - 1)}{2}$$

Where, T= number of datasets = a+b+c….+n; V_i = number of releases of i_{th} project.

For example, the number of releases for A project is a. Thus, based on the above formula, the total possible combinations in this work are 114.

9.4.2 Measuring Distributional Characteristics

For the 12 different datasets, various descriptive statistics are calculated which measure the distributional characteristics of each independent variable (number of independent variables are 7). We have measured 11 distributional characteristics (indicators) for each variable. As discussed, we have 114 possible train-test combinations. Thus, for each datapoint, a combination of 144 attributes is created as follows:

1. The distributional characteristics of each variable of training set are combined to get 77 (7 independent variables *11indicators) attributes.

2. The distributional characteristics of each variable of testing set are combined

to get 77 (7 independent variables *11indicators) attributes.

3. 77 attributes of training set and 77 attributes of testing set are further combined to yield a total of 144 attributes.

The various indicators that we used to measure the distributional characteristics are mode, median, mean (to describe the central tendency of attributes), standard deviation, variance, skewness, kurtosis, minimum, maximum, sum, first quartile and third quartile.

9.4.3 Predicting Model for Each Possible Prediction

We have used LB to build the model for each valid train - test pair (A, B). We predicted the model using the dataset A and tested it on the dataset B. Thus, in this study, we have 114 model predictions. For each prediction, we say whether the prediction is valid or not valid based on the values of performance measures, precision and AUC. If AUC is greater than 0.65 and precision is greater than 50%, we have considered the prediction as valid or successful. This we call as the 'result' of each prediction, which is discrete or binary in nature. In this study, for each prediction, we have defined a datapoint having three parts: 1) the first part consists of distributional characteristics of training set, 2) the second part consists of distributional characteristics of the testing set, 3) the third part consists of the result of each prediction (i.e. valid or not valid). Thus, we have 114 datapoints, each having 145 attributes (77 number of attributes comprising of distributional characteristics of training set, 77 number of attributes comprising of distributional characteristics of testing set and 1 attribute comprising of the result of the prediction).

9.4.4 Verifing the Relationship Between Distributional Characteristics of Datasets and Selection of Training Set

One of the objectives is to determine whether the distributional characteristics of training and testing set have an impact on the cross - project predictions. The studies by Zimmermann et al. [155] and Watanabe et al. [145] show that the distributional characteristics are essential for identifying successful cross - project defect prediction. In this work, we constructed a CRT on the dataset generated above (consisting of 114 datapoints). The independent variables are the distributional characteristics of training and testing set (77+77=144 attributes) and the binary dependent variable is the result of each prediction (valid/not valid). The validation technique used is 10 - cross validation. If CRT predicts the valid predictions with high accuracy, then we conclude that the cross - project predictions have an impact on the distributional characteristics of the training and testing sets.

9.5 Result Analysis

In this section, we discuss the RQs stated in section 9.1.

RQ1: How does the accuracy of the model differs when we train the model using the different releases of the same project (i.e. performing inter - release validation) or when we use different project (i.e. performing inter - project validation)?

For each of the possible train/test combination (A, B), we constructed LB model using dataset A and tested it on dataset B. We measured the performance in terms of precision and AUC. We are interested in knowing the number of successful cross - project predictions (precision > 50% and AUC > 0.65). We observed that out of a total of 114 predictions, 41 satisfy our criteria and thus are referred as successful predictions (i.e. the success rate is 36%). This is quite high when we compare it with the success rate obtained by [155]. Zimmermann et al. [155] ran 622 cross - project predictions and found only 3.4% actually worked. Within the successful predictions, we compared the number of successful inter - project and inter - release predictions. Inter - release predictions are the predictions where the training and testing sets belong to the same project, but different releases. Whereas, in inter - project predictions, training and testing sets belong to different projects. It would seem that the different release of same project will predict each other well, as they are developed under same environment, have similar characteristics etc. However, there is a difference of opinion in this issue by previous studies. Some of the previous studies [117, 137, 146, 147] have shown that different versions from the same project have predicted each other well. However, the latest study by He et al. [73] gave unexpected result, i.e. decreased prediction accuracy for the same (ranging from 0.32% to 4.67%). In table 9.1, the number of successful inter - project and inter - release predictions are analyzed. If two datasets A and B belong to the same project, then the total possible train - test combinations (A,B : B should be later version than A) are 18 (6 combinations for each Abdera, POI and Rave). In other words, the total number of inter - release combinations is 8. Out of 18 combinations, we observed that the number of successful predictions is 12, i.e. approximately 67% success rate. On the other hand, if two datasets A and B belong to the different projects, then the total number of train - test combinations (A,B) is 96 (114-18). Thus, the number of inter-project combinations is 96. Out of 96, the number of successful predictions is 29, i.e. approximately 30% success rate. This is significantly lower than the success rate of inter - release predictions. Thus, in this work, we concluded that the inter - release predictions give higher accuracy than the inter - project predictions.

Table 9.2 shows the range of AUC for successful predictions. For Abdera as testing project, we observed that when the same project, i.e. Abdera is used as training project (different release of the same project), the range of AUC is 0.67 to 0.78. Whereas, when different projects (Poi and Rave) are used as training sets, we observed that the range of AUC is approximately the same as when the training set is Abdera. This shows that the accuracy of the inter - project and inter - release predictions is approximately the same. Same observation is concluded when the testing project is Poi and Rave.

Table 9.1: Number of Successful Predictions

TRAIN/TEST	Abdera	POI	Rave	Total
Abdera	5	5	5	15
POI	10	4	0	14
Rave	9	0	3	12
Total	24	9	8	41

Table 9.2: AUC of Successful Predictions

Testing Project	Training Project	Min. AUC	Max. AUC
Abdera	Abdera	0.67	0.78
	Poi	0.67	0.71
	Rave	0.67	0.71
Poi	Abdera	0.66	0.72
	Poi	0.67	0.69
	Rave	-	-

Testing Project	Training Project	Min. AUC	Max. AUC
Rave	Abdera	0.65	0.72
	Poi	-	-
	Rave	0.66	0.68

RQ2: Are characteristics of data sets valuable for selecting the suitable training data for cross - project change prediction?

We generated a total of 114 possible predictions and the number of successful predictions is 41. To determine whether the characteristics of the datasets help in selecting the suitable training set, we applied decision tree (CRT) on the dataset of 114 data-points having 145 attributes (144 attributes consisting of distributional characteristics of training and testing sets + 1 attribute for the result of prediction). CRT trained from the train/test instances consisted of 15 nodes, out of which 8 are leaf nodes. The leaf nodes contain the value of the result of prediction (dependent variable). We observed three nodes resulted in the value 'yes' from which we generated the classification rules. These classification rules can be used to select a suitable training set for a given test set. We also performed 10 - cross validation on this dataset and obtained precision as 85.6% and the AUC as 0.98. Taking into consideration the success rate for inter - project predictions (67%) and the success rate for inter - release predictions (30%), we can say that the performance of CRT is very high. Thus, we concluded that the distributional characteristics of the datasets play an important role in selecting the suitable training set for a given test set.

9.6 Discussion

Cross - project prediction is gaining wide importance for the projects with insufficient or no data to build the prediction models. In this chapter, we have used four releases each of three open source projects, Apache Abdera, POI and Rave to conduct cross - project predictions. We obtained a total 12 datasets (4*3) and made 114 total possible train - test combinations (A, B), where A is the training dataset and B is the testing dataset. For each (A,B) combination, we constructed a LB model and found that the number of successful predictions (precision > 0.5 and AUC > 0.65) is 41. Thus, we obtained a total success rate of 36%.

Among the successful predictions, we found that the success rate of inter - release predictions is 67% and the success rate of inter - project predictions is 30%. From this, we concluded that the inter - release predictions are more accurate than the inter - project predictions. We constructed a CRT decision tree to judge the suitability of characteristics of the datasets in selecting the training set for a given test set. We obtained high values of precision (85.6%) and the AUC (0.98). This shows that selection of training set depends on the distributional characteristics of the datasets. For a given test set, we can select an appropriate training set using the classification rules (that produced a ‘yes’) of CRT.

Chapter 10

Conclusion

10.1 Summary of the Work

Assessing various quality attributes during the early phases of SDLC is a promising step towards the improvement of software quality. Several quantitative models constructed using statistical methods such as simple linear discriminant analysis or LR to more complex ML techniques have been proposed in literature which can be used to predict various quality attributes. However, there is no well - accepted theory and sufficient empirical studies to conclude the most suitable technique to predict software quality attributes. In addition to constructing quantitative models to measure the quality, the threshold values (also known as risk indicators) of metrics can also be identified, assessed and validated on various OSS to predict change and fault prone classes of the software. The evaluation of the models and the thresholds can be done using various performance measures and statistical tests.

We have presented and explained in detail the research methodology followed in this thesis. The initial steps that are performed to conduct an empirical study such as identifying and describing the dependent variables (change and fault proneness), independent variables (OO metrics) and empirical data collection process have been explained in detail. Then, the metric selection procedure and the correlation analysis method are also explained. The methods used for the quality model prediction (constructing of models and identifying thresholds) followed by the techniques available for validating the models (inter - project and intra - project) are discussed. Finally, we summarized various performance measures and statistical tests used to evaluate the performance of the prediction models.

To have a fair evaluation of all the studies relevant to the area of change prediction, a systematic review has been conducted. We have systematically summarized the empirical evidence obtained from the existing literature and provided a brief description of each study in terms of the independent variables, the data analysis techniques, the performance measures and the software used. We obtained the answers to various research questions which primarily focus on the following issues: 1) use of risk indicators for predicting quality attributes; 2) use of correct performance measures and statistical tests for evaluating the performance of various models; 3)type of software repositories, metrics and learning techniques; 4) type of validation technique for validating the models and 5) significant metrics for predicting change proneness. The results of the review showed that there is lack of work on identifying the threshold of metrics and the use of inter - project validation. Also, the metrics significant in predicting change proneness could not be identified as the studies in literature have not used feature reduction techniques. From this systematic review, we identified directions and guidelines for this work such as : (1) more studies should be performed with large OSS and different ML techniques should be used to construct the models which should be evaluated using different performance measures and statistical tests, (2) the usage of thresholds for change prediction should be explored and used, (3) inter - project validation should be used to validate the models. We used and incorporated the identified future directions for quality assessment.

The OSS phenomenon significantly impacts both the public and private sector of software organizations [70].The existence of freely available software leads to faster adoption of technology, less cost and more innovation [20]. OSS is also characterized by better security, no vendor or copyright issues and higher quality. Also, OSS is developed with a different approach and methodology. Users are generally treated as co-developers and development is accomplished through collaborative effort of programmers. These advantages of OSS led to a significant adoption of open source products in various domains [112, 126]. In this research, we have explored various widely used and popular OSS for prediction of quality models. For example, the fault data is obtained using four OSS namely Ivy, Tomcat, Ant, JEdit and Sakura obtained from Promise repository and one proprietary, NASA proprietary software, KC1. Besides this, multiple popular OSS are used for change data, i.e. Android, Abdera OS, Poi, Rave, JavaTreeView, Freemind, Frinika, Xerces and Xalan.

The usage of ML techniques is becoming increasingly popular across a range of domains such as finance, medicine, engineering, geology, and physics. ML techniques are used for making predictions based on some past observations and are widely used in various classification problems Besides this, they can also be used for predictive analysis and constructing various models for predicting quality attributes. However, literature shows that ML techniques are not much used for constructing models to predict change proneness. The quality models are mostly built using the traditional statistical method, LR. Moreover, the present literature does not allow to identify a suitable ML technique which can used for model construction. Hence, more studies need to be conducted exploring the use of ML techniques for model construction.

Keeping in view the above issues, we have compared the performance of 15 data analysis techniques (14 ML and one statistical) on five official releases of Android operating system. This extensive comparison provides an opportunity to fairly evaluate all the techniques and enables to judge the performance of one technique over the other. The results are validated using 10 - cross and inter - release validation (the models derived from a release ’r’ are validated on releases ’r+1’, ’r+2’ etc.). The results concluded that bagging and RF have outperformed all the other techniques and can be used to predict change prone classes in the future or upcoming releases of Android or similar software.

In one of the other study also, we have explored the usage of various ML techniques for construction of prediction models. For empirical validation, two releases of an OSS, ‘JTreeView’ are used. Among multiple ML techniques used, bagging and RF have outperformed all the other techniques and can be used by researchers for predicting change prone classes of similar software.

Software metrics are used to measure various internal characteristics of software such as size, complexity, coupling, cohesion etc. Once the values of software metrics are determined, there should be a technique or methodology to assess if the metric values are good or bad. For this, we have identified thresholds of metrics. Identifying the thresholds of metrics allow managers to concentrate on classes where metric values exceed the threshold values as these classes may have higher complexity and require management attention. We have also identified the advantages of thresholds over metric models for predicting change and fault prone classes as stated below:

1. Thresholds allow to identify which software metric to alter. The predicted change prone classes can be redesigned by altering only the metrics whose values exceed their respective thresholds.
2. They give an exact estimate of the amount of change each metric should undergo.
3. Threshold models lead to saving of resources since training the metric models is time consuming task, and thus it is not efficient on a daily basis to use metric models.

Although many empirical studies on OO metrics have been conducted, very few have identified threshold values of these metrics and thus, suggested guidelines on their interpretation and application. In lieu of this, we have conducted a study where we computed the thresholds of various OO metrics for predicting the classes that are prone to changes. In addition to identifying the thresholds of OO metrics, the traditional metric models are also constructed so that a comparison can be drawn between the two methodologies. Thus, the purpose of this study is to evaluate and assess the proposed threshold based methodology and compare its results with the traditional model based methodology. Thresholds are calculated for the metrics of Freemind 0.9.0 and Xerces 2.9.0 using a statistical methodology based on LR. The effectiveness of the proposed threshold methodology is judged on various releases of Freemind

(Freemind 0.9.1, 1.0.0) and Xerces (2.9.1, 2.10.0). In addition to this, external validation on two different projects, Frinika 0.2.0 and Xalan 2.6.0 has also been carried out. Among a number of ML techniques used to validate the models, MLP and CRT have outperformed the other models. The performance of binary models is compared with the performance of non - binary models (models without thresholds) using a statistical test known as Wilcoxon signed rank test. The results obtained from using the statistical test showed that there is no significant difference in the performance of binary and non - binary models. This shows that the threshold methodology can be effectively utilized on upcoming or the future releases of software.

We have also considered the effect of thresholds of OO metrics on fault proneness and built predictive models based on the threshold values of the metrics used. We have used the same statistical methodology to calculate the threshold values of CK metrics [36]. The empirical validation is carried out on three software: KC1 obtained from NASA and two OSS, IVY and JEdit. To demonstrate the effectiveness of the proposed methodology, inter - project validation has also been done on three OSS, Apache Ant, Apache Tomcat and Sakura. We have validated the threshold values of Ivy on Apache software (Ant, Tomcat) and the threshold values of JEdit on Sakura software. To validate the threshold values, various ML models (BN, NB, RF, SVM and MLP) are constructed. In order to analyze the performance, the results are compared with traditional metric models. Results of validation on an OSS, Ivy concluded that the models predicted using RF and MLP should be used and the results of validation on proprietary software, KC1 concluded SVM to be the best ML model. The models predicted using other ML techniques have shown comparative results. Inter - project validation is also carried out using three other OSS, Ant, Tomcat and Sakura.

The results of the inter - project validation concluded that the proposed threshold methodology can be used on the projects of similar characteristics or similar nature.

Thus, overall from the results of all the studies, we concluded that ANN (MLP), bagging and RF have outperformed all the other ML models and can be used by the researchers and practitioners to predict change and fault prone classes of the same or the similar natured project.

We have discussed that the thresholds can be computed using a statistical approach based on LR. An alternative method to calculate the thresholds of OO metrics is use of the ROC curves. We have carried out a study in which the threshold values of metrics of Android are calculated using the ROC curves. Five official releases of Android are considered and the threshold values are calculated for each of these versions. To validate the threshold values, RF model is constructed. In addition to this, inter - release validation is also carried and we observed that the AUC obtained after validating the same release and the immediate successor release is comparable. This shows that since, the threshold values of the predecessor release can always be obtained, thus, they can be utilized to predict change prone classes in the immediate next release. In this study, we also calculated the thresholds using the statistical approach and compared the results of the two techniques. We observed that the thresholds of most of the OO metrics calculated using statistical approach are out of the observation range (i.e. negative values). Whereas, this situation can never arise if we calculate the threshold values using the methodology based on ROC. Thus, we concluded that researchers and practitioners should calculate the thresholds of various OO metrics using the said methodology based on ROC and use them to predict the change prone classes of the software.

Generally, the change prediction models are learned using the historical data of the same project and then used to predict change prone classes in the upcoming releases of same or other similar project. However, the unavailability of historical data is the possible scenario that may exist. In such a scenario, training of models has to be done using the data from some other project. We have conducted such inter - project predictions (training and testing the models using different projects) using four releases each of three open source Apache projects, Abdera, POI and Rave. We conducted both inter - project (different projects) and inter - release (different versions of same projects) predictions. We found that the success rate of inter - release (different releases of the same project) is 67% and the success rate of inter - project predictions (different projects) is 30%. From this, we concluded that inter - release predictions are more accurate than the inter - project predictions. We constructed a CRT based decision tree to judge the suitability of characteristics of the datasets in selecting the training set for a given test set. We concluded that selection of training set depends on the distributional characteristics of the datasets.

10.2 Applications of the Work

An extensive empirical validation and assessment of OO metrics is conducted to construct various prediction models which are evaluated using different performance measures and statistical tests. We conclude that the work in this thesis would allow researchers and software industry:

- To obtain a subset of metrics which are significant in identifying and assessing various quality attributes.

- To identify change and fault prone classes of software during early phases of SDLC.
- To take preventive focused actions which will lead to reduction in maintenance costs and improve quality.
- To identify and assess various risk indicators or thresholds of various OO metrics.
- To use thresholds as quality benchmarks to assess and compare different software products.
- To efficiently select a suitable training set for a given testing set based on the distributional characteristics of the training set.
- To understand the usage of various performance measures and statistical tests which will lead to efficient evaluation of the models.
- To evaluate and improve the quality of the resulting OO software processes and products.

10.3 Future Directions

Although we have used many datasets and various ML techniques in this thesis, in future studies we further plan to replicate the work carried out in this thesis on diffrent application domain datasets so that more generalized results can be produced. Replication is important in order to examine the set of same hypothesis in different contexts or with the aim to improve the experiment and validate the findings of previous experiments. Data collected through replications can be used to refute or accept well-formed theories and increase the evidence on which software practitioners and researchers can base their decisions. We also plan to explore the threshold methodology on a number of similar application domain software so that generalized thresholds of metrics can be reported. We also plan to perform cost benefit analysis of the models predicted for change proneness. One of the other dimension may be the use of evolutionary search based techniques to predict change and fault prone classes.

Bibliography

[1] ABDI, M., LOUNIS, H., AND SAHRAOUI, H. Analyzing change impact in object-oriented systems. In *32nd EUROMICRO Conference on Software Engineering and Advanced Applications* (2006), pp. 310-319, Sweden.

[2] ABREU, F. B., AND MELO, W. Evaluating the impact of object-oriented design on software quality. In *Proceedings of the 3rd International Software Metrics Symposium* (1996), pp. 90-99, Berlin.

[3] ABREU, F. B. E., AND CARAPUCA, R. Candidate metrics for object-oriented software within a taxonomy framework. *Journal of Systems and Software 26* (1994), pp. 87-96.

[4] AGGARWAL, K. K., AND SINGH, Y. *Software Engineering: Programs, Documentation, Operating Procedures*, 3rd edition ed. New Age International, (2008).

[5] AGGARWAL, K. K., SINGH, Y., KAUR, A., AND MALHOTRA, R. Empirical analysis for investigating the effect of object-oriented metrics on fault proneness: A replicated study. *Software Process: Improvement and Practice 16*, 1 (2006), pp. 39-62.

[6] Aggarwal, K. K., Singh, Y., Kaur, A., and Malhotra, R. Empirical study of object-oriented metrics. *Journal of Object Technology 5*, 8 (2006), pp. 149-173.

[7] Albrecht, A., and Gaffney, J. Software function source lines of code and development effort prediction: a software science validation. *IEEE Transactions on Software Engineering SE-9*, 6 (1983), pp. 639-648.

[8] Arisholm, E., Briand, L. C., and Foyen, A. Dynamic coupling measurement for object-oriented software. *IEEE Transactions on Software Engineering 30*, 8 (2004), pp. 491-506.

[9] Bansiya, J., and Davis, C. G. A hierarchical model for object-oriented design quality as-sessment. *IEEE Transactions on Software Engineering 28*, 1 (2002), pp. 4-17.

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

[11] Belsley, D., Kuh, E., and Welsch, R. *Regression Diagnostics: Identifying Influential Data and Sources of Collinearity*. John Wiley and Sons; USA, (1980).

[12] Bender, R. Quantitative risk assessment in epidemiological studies investigating threshold effects. *Biometrical Journal 41*, 3 (1999), pp. 305-319.

[13] BENLARBI, S., EMAM, K. E., GOEL, N., AND RAI, S. Thresholds for object-oriented measures. In *Proceedings 11th International Symposium on Software Reliability Engineering* (2000), pp. 24-38.

[14] BENLARBI, S., AND MELO, W. Polymorphism measures for early risk prediction. In *Proceedings of the 21st International Conference on Software Engineering* (1999), pp. 335-344, Los Angeles, USA.

[15] BIEMAN, J., AND KANG, B. Cohesion and reuse in an object-oriented system. In *Proceedings of the ACM Symposium on Software Reusability* (1995), pp. 259-262, Seattle, United States.

[16] BIEMAN, J. M., ANDREWS, A. A., AND YANG, H. J. Understanding change-proneness in oo software through visualization. In *Proceedings of the 11th IEEE International Workshop on Program Comprehension* (2003), pp. 44-53, Portland, USA.

[17] BIEMAN, J. M., STRAW, G., WANG, H., MUNGER, P. W., AND ALEXANDER, R. T. Design patterns and change proneness: an examination of five evolving systems. In *Proceedings of the 9th International Software Metrics Symposium* (2003), pp. 40-49.

[18] BOEHM, B. *Software Engineering Economics*. Englewood Cliffs, NJ. Prentice Hall, (1981).

[19] BOEHM, B., BROWN, J., KASPAR, H., LIPOW, M., MCLEOD, G., AND MERRITT, M. *Characteristics of Software Quality*. North Holland Publishing, Amsterdam, (1978).

[20] Bonaccorsi, A., and Rossi, C. Comparing motivations of individual programmers and firms to take part in the open source movement. *From community to businessKnowledge, Technology, and Policy 18*, 4 (2006), pp. 40-64.

[21] Breiman, L. Random forests. *Machine Learning 45*, 1 (2001), pp. 5-32.

[22] Briand, L., Daly, W., and Wust, J. Exploring the relationships between design measures and software quality. *Journal of Systems and Software 51*, 3 (2000), pp. 245-273.

[23] Briand, L., Wuest, J., and Lounis, H. Using coupling measurement for impact analysis in object-oriented systems. In *IEEE International Conference on Software Maintenance* (1999), pp. 475-482, Oxford, UK.

[24] Briand, L., Wust, J., and Lounis, H. Replicated case studies for investigating quality factors in object-oriented designs. *Empirical Software Engineering. International Journal 6*, 1 (2001), pp. 11-58.

[25] Briand, L. C., Daly, W., and Wust, J. A unified framework for cohesion measurement in object-oriented systems. *Empirical Software Engineering 3*, 1 (1998), pp. 65-117.

[26] Briand, L. C., Daly, W., and Wust, J. A unified framework for coupling measurement in object-oriented systems. *IEEE Transactions on Software Engineering 25*, 1 (1999), pp. 91-121.

[27] Briand, L. C., Devanbu, P., and Melo, W. An investigation into coupling measures for C++. In *Proceedings of the International Conference on Software Engineering* (1997), pp. 412-421, Boston, USA.

[28] Briand, L. C., Melo, W., and Wust, J. Assessing the applicability of fault-proneness models across object-oriented software projects. *IEEE Transactions on Software Engineering 28*, 7 (2002), pp. 706-720.

[29] Cartwright, M. An empirical view of inheritance. *Information and Software Technology 40*, 14 (1998), pp. 795-799.

[30] Cartwright, M., and Shepperd, M. An empirical investigation of an object-oriented software system. *IEEE Transactions on Software Engineering 26*, 8 (1999), pp. 786-796.

[31] Carvalho, A. B. D., Pozo, A., and Vergilio, S. R. A symbolic fault-prediction model based on multiobjective particle swarm optimization. *Journal of Systems and Software 83*, 5 (2010), pp. 868-882.

[32] Chaumun, M. A., Kabaili, H., Keller, R. K., and Lustman, F. A change impact model for changeability assessment in object-oriented software systems. *Science of Computer Programming 45*, 23 (2002), pp. 155-174.

[33] Cheng, J., and Greiner, R. Comparing bayesian network classifiers. In *Proceedings of the 15th conference on Uncertainty in artificial intelligence* (1999), pp. 101-108, Stockholm, Sweden.

[34] Chidamber, S., Darcy, D., and Kemerer, C. Managerial use of metrics for object-oriented software: an exploratory analysis. *IEEE Transactions on Software Engineering 24*, 8 (1998), pp. 629-639.

[35] Chidamber, S., and Kemerer, C. Towards a metrics suite for object-oriented design. In *Proceedings of the Conference on Object-Oriented Programming Systems* (1991), pp. 197-211, Arizona, USA.

[36] Chidamber, S., and Kemerer, C. A metric suite for object oriented design. *IEEE Transactions on Software Engineering 20*, 6 (1994), pp. 476-493.

[37] Coleman, D., Lowther, B., and Oman, P. The application of software maintainability models in industrial software systems. *Journal of Systems and Software 29*, 1 (1995), pp. 3-16.

[38] Cooper, G., and Herskovits, E. A bayesian method for the induction of probabilistic networks from data. *Machine Learning 9*, 4 (1992), pp. 309-347.

[39] Daly, J., Brooks, A., Miller, J., Roper, M., and Wood, M. Evaluating inheritance depth on the maintainability of object-oriented software. *Empirical Software Engineering 1*, 2 (1996), pp. 109-132.

[40] Demiroz, G., and Guvenir, A. Classification by voting feature intervals. In *Proceedings of the 9th European Conference on Machine Learning* (1997), pp. 85-92, Prague, Czech Republic.

[41] Demsar, J. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research 7* (2006), pp. 1-30.

[42] Dromey, R. G. A model for software product quality. *IEEE Transactions on Software Engineering 21*, 2 (1995), pp. 146-162.

[43] Duda, R., and Hart, P. *Pattern Classification and Scene Analysis*. John Wiley Sons, (1973).

[44] El Emam, K., Benlarbi, S., Goel, N., and Rai, S. A validation of object-oriented metrics. *Technical report, NRC/ERB 1063*, 1992.

[45] Elish, M. O., and Al-Khiaty, M. A suite of metrics for quantifying historical changes to predict future change-prone classes in object-oriented software. *Journal of Software: Evolution and Process 25*, 5 (2013), pp. 407-437.

[46] Emam, K. E., Benlarbi, S., Goel, N., Melo, W., Lounis, H., and Rai, S. N. The optimal class size for object-oriented software. *IEEE Transactions on Software Engineering 28*, 5 (2002), pp. 494-509.

[47] Emam, K. E., Benlarbi, S., Goel, N., and Rai, S. The confounding effect of class size on the validity of object-oriented metrics. *IEEE Transactions on Software Engineering 27*, 7 (2001), pp. 630-650.

[48] Emam, K. E., Melo, W., and Machado, J. The prediction of faulty classes using object-oriented design metrics. *Journal of Systems and Software 56*, 1 (2001), pp. 63-75.

[49] Erni, K., and Lewerentz, C. Applying design-metrics to object-oriented frameworks. In *Proceedings of the 3rd International Software Metrics Symposium* (1996), pp. 64-74, Berlin, Germany.

[50] Eski, S., and Buzluca, F. An empirical study on object-oriented metrics and software evolution in order to reduce testing costs by predicting change-prone classes. In *4th International Conference on Software Testing, Verification and Validation Workshops* (2011), pp. 566-571, Berlin,Germany.

[51] Farooq, S., Quadri, S., and Ahmad, N. Software measurements and metrics: role in effective software testing. *International Journal of Engineering Science and Technology 3*, 1 (2011), pp. 971-680.

[52] Fawcett, T. An introduction to ROC analysis. *Pattern Recognition Letters 27*, 8 (2006), pp. 861-874.

[53] Fenton, N., and Ohlsson, N. Quantitative analysis of faults and failures in a complex software system. *IEEE Transactions on Software Engineering 26*, 8 (2000), pp. 797-814.

[54] Fowler, M. *Refactoring – Improving the Design of Existing Code*, 1st edition ed. Addison-Wesley, (1999).

[55] French, V. A. Establishing software metric thresholds. In *9th International Workshop on Software Measurement* (1999), pp. Montreal, Canada.

[56] Freund, Y., and Schapire, R. A short introduction to boosting. *Journal of Japanese Society for Artificial Intelligence 14*, 5 (1999), pp. 771-780.

[57] Friedman, M. A comparison of alternative tests of significance for the problem of m rankings. *The Annals of Mathematical Statistics 11*, 1 (1940), pp. 86-92.

[58] Giger, E., Pinzger, M., and H.C.Gall. Can we predict types of code changes? an empirical analysis. In *Proceedings of the 4th Working Conference on Mining Software Repositories* (2012), pp. 217-226, Zurich, Switzerland.

[59] Glasberg, D., Emam, K. E., Melo, W., and Madhavji, N. Validating object-oriented design metrics on a commercial java application. *TR ERB-1080,NRC* (2000).

[60] Gondra, I. Empirical validation of object-oriented metrics for predicting fault proneness models. *The Journal of Systems and Software 81*, 2 (2008), pp. 186-195.

[61] Goodman, P. *Practical Implementation of Software Metrics*. McGraw Hill, (1993).

[62] Gray, D., Bowes, D., Davey, N., Sun, Y., and Christianson, B. Further thoughts on precision. In *15th Annual Conference on Evaluation Assessment in Software Engineering* (2011), pp. 129-133, Durham, UK.

[63] Gyimothy, T., Ferenc, R., and Siket, I. Empirical validation of object-oriented metrics on open source software for fault prediction. *IEEE Transactions on Software Engineering 31*, 10 (2005), pp. 897-910.

[64] Hall, M. Correlation-based feature selection for discrete and numeric class machine learning. In *Proceeding of the 17th international conference on machine learning* (2000), pp. 359-366, Stanford, California.

[65] Halstead, M. H. *Elements of Software Science*. Elsevier North Holland, New York, (1997).

[66] Han, A. R., Jeon, S. U., Bae, D. H., and Hong, J. E. Behavioral dependency measurement for change proneness prediction in uml 2.0 design models. In *Proceedings of the 32nd IEEE International Conference on Computer Software and Applications* (2008), pp. 76-83, Turku, Finland.

[67] Han, A. R., Jeon, S. U., Bae, D. H., and Hong, J. E. Measuring behavioral dependency for improving change-proneness prediction in uml-based design models. *The Journal of Systems and Software 83*, 2 (2010), pp. 222-234.

[68] Hanley, J., and McNeil, B. The meaning and use of the area under a receiver operating characteristic ROC curve. *IEEE Transactions on Software Engineering 143*, 1 (1982), pp. 29-36.

[69] Harrison, R., Counsell, S., and Nithi, R. Experimental assessment of the effect of inheritance on the maintainability of object-oriented systems. *Journal of Systems and Software 52*, 2-3 (2000), pp. 173-179.

[70] Hauge, O., Ayala, C., and Conradi, R. Adoption of open source software in software-intensive organizations - a systematic literature review. *Information and Software Technology 52*, 11 (2010), pp. 1133-1154.

[71] Haykin, S. *Neural Networks: A Comprehensive Foundation*. Pearson Education, Delhi, (2004).

[72] He, H., and Garcia, E. Learning from imbalanced data. *IEEE Transactions on Knowledge and Data Engineering 21*, 9 (2008), pp. 1263-1284.

[73] He, Z., Shu, F., Yang, Y., Li, M., and Wang, Q. An investigation on the feasibility of cross-project defect prediction. *Automated Software Engneering 19*, 2 (2012), pp. 167-199.

[74] Henderson-Sellers, B. *Object-Oriented Metrics: Measure of Complexity*. Prentice Hall, (1996).

[75] Henry, S., and Kafura, D. Software structure metrics based on information flow. *IEEE Transactions on Software Engineering SE-7*, 5 (1981), pp. 510-518.

[76] Hosmer, D., and Lemeshow, S. Wiley; New York.

[77] Jureczko, M., and Madeyski, D. Towards identifying software project clusters with regard to defect prediction. In *Proceedings of the 6th International Conference on Predictive Models in Software Engineering* (2010), pp. Timisoara, Romania.

[78] Jureczko, M., and Spinellis, D. Using object-oriented design metrics to predict software defects. In *Proceedings of 5th international Conference on Dependability of Computer Systems, Monographs of System Dependability* (2010), pp. 69-81, Wroclaw, Poland.

[79] Kanmani, S., Uthariaraj, V. R., Sankaranarayanan, V., and Thambidurai, P. Object-oriented software prediction using neural networks. *Information and Software Technology 49*, 5 (2007), pp. 483-492.

[80] Kitchenham, B. A. Guidelines for performing systematic literature review in software engineering. *Technical report, EBSE-2007-001*, 2007.

[81] Koru, A. G., and Liu, H. Identifying and characterizing change-prone classes in two large-scale open-source products. *The Journal of Systems and Software 80*, 1 (2007), pp. 63-73.

[82] Koru, A. G., and Tian, J. Comparing high-change modules and modules with the highest measurement values in two large-scale open-source products. *IEEE Transactions on Software Engineering 31*, 8 (2005), pp. 625-642.

[83] Kubat, M., and Matwin, S. Addressing the curse of imbalanced training sets: one-sided selection. In *Proceeding of the 14th International conference on machine learning* (1997), pp. 179-186, Nashville, USA.

[84] Landwehr, N., Hall, M., and Frank, E. Logistic model trees. *Machine Learning 95*, 12 (2005), pp. 161-205.

[85] Lee, Y., Liang, B., Wu, S., and Wang, F. Measuring the coupling and cohesion of an object-oriented program based on information flow. In *Proceedings of the International Conference on Software Quality* (1995), pp. 81-90, Maribor, Slovenia.

[86] Lessmann, S., Baesens, B., Swantje, C., and Pietsch. Benchmarking classification models for software defect prediction: a proposed framework and novel findings. *IEEE Transactions on Software Engineering 34*, 4 (2008), pp. 485-496.

[87] Li, W. Another metric suite for object-oriented programming. *Journal of Systems and Software 44*, 2 (1998), pp. 155-162.

[88] Li, W., and Henry, W. Object-oriented metrics that predict maintainability. *Journal of Software and Systems 23*, 2 (1993), pp. 111-122.

[89] Lindvall, M. Are large C++ classes change-prone? An empirical investigation. *Software Practice and Experience 28*, 15 (1998), pp. 1551-1558.

[90] Lindvall, M. Measurement of change: stable and change-prone constructs in a commerical C++ system. In *Proceedings of the 6th International Software Metrics Symposium* (1999), pp. 40-49, Boca Raton, FL, USA.

[91] Lorenz, M., and Kidd, J. *Object-Oriented Metrics*. Prentice Hall, (1994).

[92] Lu, H., Zhou, Y., Xu, B., Leung, H., and Chen, L. The ability of object-oriented metrics to predict change-proneness: a meta-analysis. *Empirical Software Engineering 17*, 3 (2011), pp. 200-242.

[93] Lyu, M. R. *Handbook of Software Reliability Engineering*. IEEE Computer Society Press and McGraw-Hill, (1996).

[94] Ma, Y., and Cukic, B. Adequate and precise evaluation of quality models in software engineering studies. In *Proceedings of the 3rd International Workshop Predictor Models in Software Engineering* (2007), pp. 1-9, Minneapolis, USA.

[95] Malhotra, R. *Empirical Research in Software Engineering: Concepts, Analysis, and Applications*. CRC Press, (2015).

[96] MALHOTRA, R., AND AGRAWAL, A. Cms tool: calculating defect and change data from software project repositories. *ACM SIGSOFT Software Engineering Notes 39*, 1 (2014), pp. 1-5.

[97] MALHOTRA, R., AND BANSAL, A. Software change proneness prediction: A literature review (to be published). *International Journal of Computer Applications in Technology*.

[98] MALHOTRA, R., AND BANSAL, A. Cross project change prediction using open source projects. In *3rd International Conference on Advances in Computing, Communications and Informatics* (2014), pp. 201-207, New Delhi, India.

[99] MALHOTRA, R., AND BANSAL, A. Predicting software change in an open source software using machine learning algorithms. *International Journal of Reliability Quality and Safety Engineering 20*, 6 (2014), pp. 1-14.

[100] MALHOTRA, R., AND BANSAL, A. Fault prediction considering threshold effects of object oriented metrics. *Expert Systems 32*, 2 (2015), pp. 203-219.

[101] MALHOTRA, R., AND BANSAL, A. Predicting change using software metrics : a review. In *4th International Conference on Reliability, Infocom Technologies and Optimization* (2015), pp. -, New Delhi, India.

[102] MALHOTRA, R., AND BANSAL, A. Prediction of change prone classes using threshold methodology. *Advances in Computer Science and Information Technology 2*, 11 (2015), pp. 30-35.

[103] Malhotra, R., and Khanna, M. Investigation of relationship between object-oriented metrics and change proneness. *Machine Learning and Cybernetics 4*, 4 (2013), pp. 273-286.

[104] Malhotra, R., and Khanna, M. A new metric for predicting software change using gene expression programming. In *Proceedings of the 5th International Workshop on Emerging Trends in Software Metrics* (2014), pp. 8-14, Hyderabad, India.

[105] Malhotra, R., and Peer, A. Application of adaptive neuro-fuzzy inference system for predicting software change proneness. In *International Conference on Advances in Computing Communications and Informatics* (2013), pp. 2026-2031, Mysore, India.

[106] Malhotra, R., Pritam, N., Nagpal, K., and Upmanyu, P. Defect collection and reporting system for git based open source software. In *International Conference on Data Mining and Intelligent Computing* (2014), pp. 1-7, New Delhi, India.

[107] Marinescu, C. How good is genetic programming at predicting changes and defects? In *16th International Symposium onSymbolic and Numeric Algorithms for Scientific Computing* (2014), pp. 544-548, Timis, Romania.

[108] Mccabe, T. J. A complexity measure. *IEEE Transactions on Software Engineering SE-2*, 4 (1976), pp. 308-320.

[109] McCall, J., Richards, P., and Walters, G. Factors in software quality. *National Technical Information Service 1,2 and 3* (1977).

[110] Menzies, T., Greenwald, J., and Frank, A. Data mining static code attributes to learn defect predictors. *IEEE Transactions on Software Engineering 33*, 1 (2007), pp. 2-13.

[111] Michalak, K., and Kwasnicka, H. Correlation-based feature selection strategy in neural classification. In *Sixth international conference on intelligent systems design and applications* (2006), pp. 741-746, Jinan, China.

[112] Morgan, L., and Finnegan, P. How perceptions of open source software influence adoption: an exploratory study. In *Proceedings of the 15th European Conference on Information Systems* (2007), pp. 973-984, St. Gallen, Switzerland.

[113] Myrtveit, I., Stensrud, E., and M, S. Reliability and validity in comparative studies of software prediction models. *IEEE Transactions on Software Engineering 31*, 5 (2005), pp. 380-391.

[114] Nagappan, N., Ball, T., and Zeller, A. Mining metrics to predict component failures. In *Proceedings of the 28th International Conference on Software Engineering* (2006), pp. 452-461.

[115] Neapolitan, R. E. *Learning Bayesian Networks*. Prentice Hall, (2004).

[116] Nejmeh, B. A. Npath: A measure of execution path complexity and its application. *Commun. ACM 31*, 2 (1988), pp. 188-200.

[117] Ostrand, T., Weyuker, E., and Bell, R. Predicting the location and number of faults in large software systems. *IEEE Transactions on Software Engineering 31*, 4 (2005), pp. 340-355.

[118] Pai, G. Empirical analysis of software fault content and fault proneness using Bayesian methods. *IEEE Transactions on Software Engineering 33*, 10 (2007), pp. 675-686.

[119] Penta, M. D., Cerulo, L., Gueheneuc, Y. G., and Antonio, G. An empirical sudy of the relationships between design pattern roles and class change proneness. In *IEEE International conference on Software Maintenance* (2008), pp. 217-226, Beijing, China.

[120] Posnett, D., Bird, C., and Devanbu, P. An empirical study on the influence of pattern roles on change-proneness. *Empirical Software Engineering 16*, 3 (2011), pp. 396-423.

[121] Prechelt, L., Unger, B., Philippsen, M., and Tichy, W. A controlled experiment on inheritance depth as a cost factor for code maintenance. *J. Systems and Software 65*, 2 (2003), pp. 115-126.

[122] Qinlan, J. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, (1993).

[123] R.Malhotra, and A.Jain. Fault prediction using statistical and machine learning methods for improving software quality. *Journal of Information Processing Systems 8*, 2 (2012), pp. 241-262.

[124] Romano, D., and Pinzger, M. Using source code metrics to predict change-prone java interfaces. *Technical Report, ISSN 1872-5392*, 2011.

[125] Rosenberg, L. Metrics for object-oriented environment. In *Proceedings of EFAITP/AIE Third Annual Software Metrics Conference* (1997).

[126] Scacchi, W., Feller, J., Fitzgerald, B., Hissam, S., and Lakhani, K. Understanding free/open source software development processes. *Software Process: Improvement and Practice 11*, 2 (2006), pp. 95-105.

[127] Sharafat, A. R., and Tahvildari, L. A probabilistic approach to predict changes in object-oriented software system. In *11th European Conference on Software Maintenance and Reengineering* (2007), pp. 27-38, Amsterdam, Netherlands.

[128] Sharafat, A. R., and Tahvildari, L. Change prediction in object-oriented software systems: a probabilistic approach. *Journal of Software 3*, 5 (2008), pp. 26-39.

[129] Shatnawi, R. A quantitative investigation of the acceptable risk levels of object-oriented metrics in open-source systems. *IEEE transactions on Software Engineering 36*, 2 (2010), pp. 216-225.

[130] Shatnawi, R., and Li, W. The effectiveness of software metrics in identifying error-prone classes in post release software evolution process. *The Journal of Systems and Software 81*, 11 (2008), pp. 1868-1882.

[131] Shatnawi, R., Li, W., Swain, J., and Newman, T. Finding software metrics threshold values using ROC curves. *Journal of Software Maintenance and Evolution: Research and Practice 22*, 1 (2010), pp. 1-16.

[132] Sherrod, P. Dtreg predictive modeling software.

[133] Simon, F., Steinbruckner, F., and Lewerentz, C. Metrics based refactoring. In *Proceedings of the 5th European Conference on Software Maintenance and Reengineering* (2001), pp. 30-38, Washington DC, USA.

[134] Singh, Y., Kaur, A., and Malhotra, R. Empirical validation of object-oriented metrics for predicting fault proneness models. *Software Quality Journal 18*, 1 (2010), pp. 3-35.

[135] Sjoberg, D., Hannay, J., Hansen, O., Kampenes, V., Karahasanovic, A., Liborg, N., and Rekdal, A. A survey of controlled experiments in software engineering. *IEEE Transactions on Software Engineering 31*, 9 (2005), pp. 733-753.

[136] Tang, M. H., Kao, M. H., and Chen, M. An empirical study on object-oriented metrics. In *Proceedings of the 6th International Software Metrics Symposium* (1999), pp. 242-249, Boca Raton, Florida.

[137] Tosun, A., Turhan, B., and Bener, A. Practical considerations in deploying ai for defect prediction: a case study within the turkish telecommunication industry. In *Proceedings of the 5th International Conference on Predictor Models in Software Engineering* (2009), pp. 1-9, Vancouver, Canada.

[138] Tsantalis, N., Chatzigeorgiou, A., and Stephanides, G. Predicting the probability of change in object-oriented systems. *IEEE Transactions on Software Engineering 31*, 7 (2005), pp. 601-614.

[139] Tu, J. Advantages and disadvantages of using artificial neural networks versus logistic regression for predicting medical outcomes. *Journal of Clinical Epidemiology 49*, 11 (1996), pp. 1225-1231.

[140] Turhan, B., Bener, A., and Menzies, T. Regularities in learning defect predictors. In *The 11th International Conference on Product Focused Software Development and Process Improvement* (2010), pp. 116-130, Limerick, Ireland.

[141] Turhan, B., Menzies, T., and Bener, A. On the relative value of cross-company and within company data for defect prediction. *Empirical Software Engineering 14*, 5 (2009), pp. 540-578.

[142] Ulm, K. A statistical method for assessing a threshold in epidemiological studies. *Statistics in Medicine 10*, 3 (1991), pp. 341-349.

[143] Wahyudin, D., Ramler, D., and Biffl, S. A framework for defect prediction in specific software project contexts. In *The 3rd IFIP Central and East European Conference on Software Engineering Techniques* (2008), pp. 295-308, Brno, Czech Republic.

[144] Wallmuller, E. *Software Quality Assurance: A Quality Approach*. Prentice-Hall International, (1994).

[145] Watanabe, S., Kaiya, H., and Kaijiri, K. Adapting a fault prediction model to allow inter language reuse. In *Proceedings of the International Workshop on Predictive Models in Software Engineering* (2008), pp. 19-24, Leipzig, Germany.

[146] Weyuker, E., Ostrand, T., and Bell, R. Do too many cooks spoil the broth? using the number of developers to enhance defect prediction models. *Empirical Software Engineering 13*, 5 (2008), pp. 539-559.

[147] Weyuker, E., Ostrand, T., and Bell, R. Comparing the effectiveness of several modeling methods for fault prediction. *Empirical Software Engineering 15*, 3 (2009), pp. 277-295.

[148] Wilcoxon, F. Individual comparisons by ranking methods. *Biometrics 1*, 6 (1945), pp. 80-83.

[149] Yu, P., Systa, T., and Muller, H. Predicting fault-proneness using oo metrics: an industrial case study. In *Proceedings of the 6th European Conference on Software Maintenance and Reengineering* (2002), pp. 99-107, Budapest, Hungary.

[150] Zhang, H. The optimality of naive bayes. In *Proceedings of the 17th International Florida Artificial Intelligence Research Society Conference* (2004), pp. 562-567, Florida.

[151] Zhang, H., and Zhang, X. Comments on data mining static code attributes to learn defect predictors. *IEEE Transactions on Software Engineering 33*, 9 (2007), pp. 635-637.

[152] ZHOU, Y., AND LEUNG, H. Empirical analysis of object-oriented design metrics for predicting high and low severity faults. *IEEE Transactions on Software Engineering 32*, 10 (2006), pp. 771-789.

[153] ZHOU, Y., LEUNG, H., AND XU, B. Examining the potentially confounding effect of class size on the associations between object oriented metrics and change proneness. *IEEE Transactions on Software Engineering 35*, 5 (2009), pp. 607-623.

[154] ZHOU, Y., XU, B., AND LEUNG, H. On the ability of complexity metrics to predict fault-prone classes in object-oriented systems. *Journal of Systems and Software 83*, 4 (2010), pp. 660-674.

[155] ZIMMERMANN, T., NAGAPPAN, N., AND GALL, H. Cross-project defect prediction:a large scale experiment on data vs. domain vs. process. In *Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering* (2009), pp. 91-100, Amsterdam, Netherland.

[156] ZUSE, H. *Software Complexity: Measures and Methods*. Walter De Gryter, Berlin, (1990).

SUPERVISOR'S BIOGRAPHY

Ruchika Malhotra

Associate Head and Assistant Professor

Department of Software Engineering,

Delhi Technological University, Delhi

Email: ruchikamalhotra2004@yahoo.com

EDUCATIONAL QUALIFICATIONS:

Ph.D. (Information Technology), MCA(SE), BIS(H)

Prior to joining the University, she was an assistant professor at the University School of Information Technology, Guru Gobind Singh Indraprastha University, Delhi, India. She received her master's and doctorate degree in software engineering from the University School of Information Technology, Guru Gobind Singh Indraprastha

University, Delhi, India. She has been awarded the prestigious UGC Raman Postdoctoral Fellowship by the Indian government for pursuing postdoctoral research from the Department of Computer and Information Science, Indiana University-Purdue University Indianapolis (2014–15), Indianapolis, Indiana. She received the prestigious IBM Faculty Award 2013. She has received the Best Presenter Award in Workshop on Search Based Software Testing, ICSE, 2014, Hyderabad, India. She is an executive editor of Software Engineering: An International Journal and is a coauthor of a book Object Oriented Software Engineering. Her research interests are in empirical research in software engineering, improving software quality, statistical and adaptive prediction models, software metrics, the definition and validation of software metrics, and software testing. Her H-index as reported by Google Scholar is 17. She has published more than 100 research papers in international journals and conferences. She has visited foreign Universities like Imperial College, London, UK; Indiana University-Purdue University Indianapolis, Indianapolis, Indiana; Ball State University, Muncie, Indiana; and Harare Institute of Technology, Zimbabwe. She has served on the technical committees of several international conferences in the area of software engineering.

AUTHOR'S BIOGRAPHY

Ankita Bansal

Assistant Professor, Department of Information Technology
Netaji Subhas Institute of Technology, Sector 3, Dwarka, Delhi 110078
Email: ankita.bansal06@gmail.com

Thesis Title:
Development of Techniques and Models for Improving Software Quality

Educational Qualifications:
Master of Engineering (Computer Technology and Applications), Bachelor of Technology (Computer Science)

Academic Achievements:
Secured 1st position in order of merit in M.E. and awarded a scholarship of Rs. 10,000 in each semester of M.E. for the same from Delhi Technological University (formerly Delhi College of Engineering)

Research Interests:
Software quality, software metrics, software testing, development and validation of prediction models