CAPITAL UNIVERSITY OF SCIENCE AND
TECHNOLOGY, ISLAMABAD

Predicting the Number of Software Faults Using Significant Process and Code Metrics

by

Sarmad Hassan

A thesis submitted in partial fulfillment for the
degree of Master of Science

in the

Faculty of Computing
Department of Computer Science

2020

This thesis is devoted to My Parents. I have a special feeling of gratitude for my beloved parents, siblings and friends. Special thanks to my supervisor whose uncountable confidence enabled me to reach this milestone.

CERTIFICATE OF APPROVAL

Predicting the Number of Software Faults Using Significant Process and Code Metrics

by

Sarmad Hassan

(MCS171035)

THESIS EXAMINING COMMITTEE

S. No.	Examiner	Name	Organization
(a)	External Examiner	Dr. Basit Shahzad	NUML, Islamabad
(b)	Internal Examiner	Dr. Nayyer Masood	CUST, Islamabad
(c)	Supervisor	Dr. Aamer Nadeem	CUST, Islamabad

Dr. Aamer Nadeem
Thesis Supervisor
December, 2020

Dr. Nayyer Masood
Head
Dept. of Computer Science
December, 2020

Dr. Muhammad Abdul Qadir
Dean
Faculty of Computing
December, 2020

Author's Declaration

I, **Sarmad Hassan** hereby state that my MS thesis titled "**Predicting the Number of Software Faults Using Significant Process and Code Metrics**" is my own work and has not been submitted previously by me for taking any degree from Capital University of Science and Technology, Islamabad or anywhere else in the country/abroad.

At any time if my statement is found to be incorrect even after my graduation, the University has the right to withdraw my MS Degree.

(Sarmad Hassan)
Registration No: MCS171035

Plagiarism Undertaking

I solemnly declare that research work presented in this thesis titled "**Predicting the Number of Software Faults Using Significant Process and Code Metrics**" is solely my research work with no significant contribution from any other person. Small contribution/help wherever taken has been duly acknowledged and that complete thesis has been written by me.

I understand the zero tolerance policy of the HEC and Capital University of Science and Technology towards plagiarism. Therefore, I as an author of the above titled thesis declare that no portion of my thesis has been plagiarized and any material used as reference is properly referred/cited.

I undertake that if I am found guilty of any formal plagiarism in the above titled thesis even after award of MS Degree, the University reserves the right to withdraw/revoke my MS degree and that HEC and the University have the right to publish my name on the HEC/University website on which names of students are placed who submitted plagiarized work.

(Sarmad Hassan)
Registration No: MCS171035

List of Publications

It is certified that following publication(s) have been made out of the research work that has been carried out for this thesis:-

1. **Predicting the number of software faults: Investigation of significant process and code metrics, IET journal (submitted for publication)**

(Sarmad Hassan)
Registration No: MCS171035

Acknowledgements

First of all I am really grateful to Allah, the Almighty for empowering me the courage to complete my thesis. I would like to render my heartfelt regards to my supervisor, **Dr. Aamer Nadeem** his continuous guidance made this research possible. His advice and friendly demeanour not only made him accessible but also encourage me to discuss all sort of difficulties in a rather warm environment. I highly appreciate his patience and expertise. I would also like to thank all members of Center for Software Dependability (CSD) for their valuable suggestions. I am grateful to Dr. Shahid Iqbal who helped me through this journey.

My deepest gratitude goes to my beloved parents and siblings for their encouragement. A very special thanks to my mother for her love and moral support.

Last but not the least and without hesitation, I would like to thank myself, for surviving all the stress.

(Sarmad Hassan)

Registration No: MCS171035

Abstract

Software fault prediction is used to predict faults and defects in a software. It is a basic step in software quality assurance process. The prediction of faults depends upon the software metrics which have several types. Most common type of software metrics are code metrics and process metrics. Calculating software metrics from a software and then using them to predict faults in software is an expensive process in terms of computation power and time. Reducing the amount of software metrics to use only important metrics can help perform fault prediction process efficiently while using fewer resources. Previous studies also show that the regression problems in SFP are not investigated as much as classification problems and process metrics are also rarely used as compared to code metrics. In our study, we have addressed this issue by selecting most influential code and process metrics for prediction of number of software faults. Wrapper based feature selection method is applied on five public datasets having code and process metrics as features and the obtained subsets of both types are added to form hybrid metrics set. Then this hybrid metrics set is used to evaluate and predict number of faults in software by applying random forest as a ML model. Different other types of evaluations were also performed which include metric type (process and code) comparisons, category wise comparison and individual metric comparison. The results show that selected hybrid set obtained using wrapper subset selection performed very well as compared to all metrics set for predicting number of faults and process metrics performed better than code metrics.

Key Words: software fault prediction, software defect prediction, process metrics, code metrics, number of faults, machine learning, feature selection paths, control flow graph

Contents

List of Figures

List of Tables

Abbreviations

AAE	Average Absolute Error
AUC	Area Under the Curve
CART	Classification and Regression Technique
CBO	Coupling Between Objects
CK	Chidamber Kemerer
CPDP	Cross Project Defect Prediction
DIT	Depth of Inheritance Tree
FeSCH	Feature Selection using Clusters of Hybrid data
FS	Feature Selection
IDE	Integrated Development Environment
LCOM	Lack of Cohesion of Methods
LCQ	Lack of Coding Quality
LOC	Lines of Code
ML	Machine Leaning
MSE	Mean Squared Error
NOC	Number Of Children
RFC	Response For Class
RRSE	Root Relative Squared Error
RF	Random Forest
SFP	Software Fault Prediction
SMOTE	Synthetic Minority Oversampling Technique
SQA	Software Quality Assurance
WMC	Weighted Methods per Class

Chapter 1

Introduction

Software fault prediction is applied to identify faulty modules in a software. A software usually contains several faults and bugs when it is first coded. Software developers apply different fault prediction techniques to identify and remove these faults before and after the software is available to end user. This process continues until all known faults and bugs are dealt with and it is termed as software quality assurance (SQA) [1].

Faults or bugs in a software can render it useless and inconvenient for end users. In severe cases, it can lead to important data loss and/or failure of software functionality. The faulty modules that are identified using fault prediction techniques are analyzed and the predicted faults are removed by software developers. This way the quality and reliability of the software is improved. Thus, making it trustworthy and easier to use for end users.

Software fault prediction is carried out using different types of software metrics. These software metrics are invoked as features in training different types of machine learning techniques. The output of these machine learning models is prediction of software faults. Depending on the type of outcome of ML models, the fault prediction process can be divided into two categories. The most common type of SFP is classification of faulty modules. In classification, the output of ML model is binary class which labels that software module as faulty or non-faulty. In

this way, the modules labeled as faulty are identified and edited/improved. In order to perform classification of software faults, appropriate classification machine learning techniques are applied as ML models.

The second type of fault prediction is regression. In regression, the output of ML model gives the number of faults or bugs in the corresponding software module. In this way, the module is not only classified as faulty but the number of faults is also identified. Having knowledge of higher and lower fault count in software modules is important as the developers can target the modules with more faults on priority. This way efficiency and better utilization of resources can be achieved which lead to better and rapid software quality improvement. Like classification, selection of ML models is also important in regression.

1.1 Software Metrics

The software metrics have a history of evolution over a period of time. Among software metrics, code metrics were one of the pioneer software metrics. Basic code metrics like McCabe's metrics were created in 1970s [2]. Most of the code metrics we use these days were created between 1980s to early 2000s. Researchers are still coming up with new software metrics to date.

Software metrics can be divided into several types based on the structure and composition of software, coding details, history of changes and authors or developers involved. There are many other types of metrics which are not commonly used in software fault prediction e.g. network metrics. The metrics which are used commonly for software fault prediction are code metrics. Most of the researchers used variable combinations of code metrics for performing evaluation related to fault prediction. Code metrics are composed of software metrics which are defined using structure of code, hierarchy of classes, size of code, inheritance, coupling of classes and cohesion among class modules etc. There also exist other types of software metrics like process metrics which are based on changes to a software over time, age of software, number of developers and authors, lines of codes added or deleted

etc. The process metrics are not as commonly used as code metrics for software fault prediction. Although many researchers have used them in their studies, the ratio of usage of process metrics as compared to code metrics is small. Many researchers have claimed process metrics to be more informative and meaningful as well as better predictors of software faults.

1.1.1 Code Metrics

The code metrics are most commonly used by researchers to perform different type of experiments and analyses in their researches. The reason for wide acceptance is their proven fault prediction performance and utility as well as easy availability of datasets containing code metrics. Some common code metrics are object-oriented metrics [3], Chidamber-Kemerer also known as CK metrics [4], Halstead metrics [5], McCabe's complexity metrics [2] and Lines of code (LOC). Moreover, some metrics introduced later are coupling metrics [6], cohesion metrics [7] and a better version of object oriented metrics in metrics suite [8]. Object oriented metrics compiled by [3] contained metrics which measure inheritance, encapsulation and polymorphism etc. The CK metrics contain metrics related to object-oriented programming like number of children (NOC), depth of inheritance tree (DIT) and response for class (RFC) etc. Halstead metrics reflect the complexity measures of code like number of operators and operators used in a program. McCabe's metrics suite contains metrics related to complexity of software and LOC is the count of coded lines in a software program. The coupling and cohesion metrics include metrics like coupling between objects (CBO), depth of inheritance tree (DIT), methods calling current method (fanIn), method calls by current method (fanOut) and public/private attributes of classes.

1.1.2 Process Metrics

Process metrics are also known as change or developer metrics. This type of metrics defines the changes occurring to a software along its development and

entities involved in changes. Some commonly known process metrics are number of revisions, code churn [9], age of software [10], lines of code added or deleted and number of authors/developers. More detailed information about metrics used in our research is given in features section.

Software metrics play vital role in software fault prediction. Acquiring these metrics and utilizing them to resolve bugs/issues in a software is a costly process and require many resources including computation power, human resources and time. By targeting higher fault count areas of software on priority, developers can make the software quality assurance method more effective, efficient and less resource demanding [11].

Many scientists and researchers have investigated software fault prediction by evaluating and using different machine learning models for fault prediction, comparing effect of different type of software metrics on fault prediction, effectiveness of performance measures used in SFP etc.

To our knowledge, a huge portion of research done in this domain is focused on classification of defect and usage of code base software metrics for prediction. The impact of different process and change metrics in fault prediction is not as much researched as of code metrics. Furthermore, the regression methods in SFP for count models of faults or defects is investigated by fewer and even those few have used code metrics for the purpose.

In our research, we investigated the impact of process metrics on software fault prediction. We have identified most important process and code metrics using wrapper subset selection for calculation of number of faults in a software.

We combined the collected smaller subsets of process and code metrics into hybrid metrics set and compared results of hybrid feature set with all feature set. We also studied the effectiveness of categories of process and code metrics as well as that of single metrics selected in our study.

To summarize our contribution in this research we have compiled some research questions which are given in section 1.4.

1.2 Problem Statement

Researchers have been trying to evaluate different types of metrics to find best and most accurate combinations of metrics for software fault prediction. Also, they have been trying to minimize the number of software metrics used for fault prediction because they are expensive to obtain and use for prediction in terms of computation, manpower and time. From literature review, we found three issues in software fault prediction which were not investigated thoroughly. The first is the use of process metrics in SFP which were not investigated as much as code metrics. The second is the studies on number of software faults (regression problem) which are also fewer as compared to classification problems. The third is the evaluation of combined metrics (code and process) for fault prediction and selection of significant code and process metrics for this purpose. We have tried to combine and address all these issues in our research.

1.3 Research Questions

RQ1: What is the performance of process metrics as compared to code metrics in software fault prediction?

To answer this question, we have compared both types of metrics using machine learning model and results are presented in chapter 4.

RQ2: Which process and code metrics subsets are most important in terms of performance for prediction of number of faults?

To answer this question, we have listed the process and code metrics selected using feature selection. We combine these two types into single hybrid metric set and also present their performance evaluation in results section of chapter 4.

RQ3: What is the effectiveness of different categories and individual selected process and code metrics?

To answer this particular research question, we divide selected process and code metrics selected through feature selection into categories and investigate performance of these categories with each other as well as performance of individual selected metrics.

1.4 Contributions

1. First of all, we have identified a most significant subset of process and code metrics using wrapper selection method for prediction of number of faults.
2. After that, we have designed an effective software faults prediction framework by using hybrid feature set and random forest as machine learning model.
3. In the next step, we have divided the selected process and code metrics into categories depending on the type of metrics.
4. At the end, our experimental results prove that selected process metrics outperformed the selected code metrics.
5. We have found set of seven most important process metrics and set of eleven most important code metrics for predicting the number of faults.

1.5 Thesis Organization

The rest of study is organized into chapters as follows;

- Chapter 2 surveys the existing work on software fault prediction.
- Chapter 3 discusses the proposed approach and its details.
- Chapter 4 presents the experimental results of proposed approach.
- Chapter 5 concludes the research and furthermore gives some future research directions.

Chapter 2

Literature Review

Chapter 1 provides details on consequences that guide us to define the problem statement. This chapter focuses on critical analysis of all the state-of-the-art approaches, as every research study is dependent on the preceding study, that have already been performed in this area. Lots of researchers have studied and focused on software fault prediction. A broad variety of literature on software fault prediction is available which uses various machine learning techniques and software metrics to predict fault count and fault proneness in a software module. To analyze and enhance the SFP method, researchers tested various types of software metrics and machine learning models.

We have compiled research papers in literature review section which are relevant to our research. These papers can be divided into two basic categories which belong to code metrics usage and process metrics usage. We have also separated papers which applied feature selection techniques to enhance software fault prediction process. These categories are given as follows:

1. Fault prediction by code metrics.
2. Fault prediction by process metrics.
3. Feature selection for SFP.

2.1 Fault Prediction by Code Metrics

Code metrics were mostly used for fault prediction studies because of easy availability of code metrics based datasets. Most common datasets used in related studies belong to PROMISE repository which can be accessed publicly worldwide. Following are some papers that experimented using code metrics and are related to our study.

Tibor Gyimothy [12] tested the values of chidamber-kamerer metrics against the number of faults utilizing various machine learning models to determine the effectiveness of CK metrics in the Mozilla and Bugzilla datasets for fault detection. The conclusion was that CBO has given the best prediction performance while the LOC metric was also good, DIT metric produced insignificant results, and NOC should not be used to predict fault.

The relationship between the complexity metrics and the number of defects was studied by Zimmermann et al. [13]. They experimented using datasets from the eclipse releases 2.0, 2.1 and 3.0. They used the spearman linear regression as a machine learning model for estimating the number of defects and used spearman correlation between the complexity metrics and the number of defects to compare relation between them. They concluded that the complexity metrics can be used to predict defects and the number of defects will be high when the code is more complex.

Khoshgoftaar et al. [14] presented assessment of five count models for predicting the number of faults. To build fault count models for two industrial software systems, they used different complexity and object-oriented metrics. Their research reveal that zero-inflated and hurdle models of negative binomial regression worked stronger for fault estimation than other count models.

Zhang et al. [15] examined the interaction between LOC and software defects. They proposed that one can use defects measured from a limited number of big software components to reliably forecast defects in general. Spearman correlation ranking was used to calculate association between LOC and faults in three variants

of eclipse and NASA datasets. Five machine learners were used to identify LOC (range) as defective / non-defective. They concluded that lines of code is an essential parameter for software defect prediction and can be used to construct reliable defect prediction models.

Marian et al. [16] assessed the effectiveness of code metrics against software fault count. They designed a Metric Calculation Tool (CKJM) and used it to extract lines of code and complexity metrics from eleven datasets to determine the relationship between metrics and number of faults. They used Pearson's correlation to explore the correlation between software metrics and the number of defects. They deduced that significant evaluation and testing cost can be saved by evaluating classes with a larger number of defects.

Erturk and Sezer [17] have implemented a Fuzzy Inference system prediction model. Their model utilizes specialist expertise and details on previous iterations to start learning when data is not present and proceeds to use a data dependent method when adequate data is available for the classification of defects. They used PROMISE datasets to build and test their models which were composed of code metrics. They used area under the curve (AUC) as Their performance measure. They concluded that their proposed prediction method can be applied to successfully predict software faults.

Fagundes et al. [18] applied five regression approaches to predict number of faults in a software. They created two step model that initially classify the results as faulty/non faulty and then predict the number of faults if results are found to be faulty. Moreover, they used NASA datasets having lines of code, complexity and cohesion metrics. They reported that their method surpassed other methods used in their research.

Rathore et al. [19] predicted number of software defects using ensemble approach based on three regression methods. They used fifteen PROMISE datasets to train and test their models which were composed of code metrics. Their results showed that in comparison to single learners, ensemble methods performed very well for number of faults prediction.

The ROC framework was used by Shatnawi [20] to define a threshold for software modules classification into faulty and non-faulty. They used dataset which contained code metrics to build and evaluate their model. They also examined imbalance and the selection of features using ROC analysis.

In one other paper, Rathore and Kumar [21] explored the ensemble methods for calculating the number of faults in a software, as they suggested that the use of ensemble methods in such a setting has not been tested. Five different learners were used to build and evaluate ensemble. They used AAE and ARE as performance metrics to report and compare results. They used eleven PROMISE datasets which were based on code metrics for evaluation. They deduced that compared to individual models, the ensemble method has shown improved performance for prediction of number of defects.

The significance of inheritance based metrics for software defect classification has been explored by Rashid et al. [22]. For this reason, they used sixty-five publicly available datasets having CK and inheritance metrics. The testing models were divided into two categories, one with inheritance and CK metrics and another with inheritance metrics only. For evaluation purposes they used artificial neural networks and used five evaluation metrics to present results. The findings indicate that the inheritance metrics make an important contribution to the software fault prediction.

Rizwan et al. [23] assessed the importance of coupling metrics for software defect classification. They investigated seven coupling metrics across eighty-seven datasets. For machine learning, they used support vector machine. Their findings demonstrate that three metrics are remarkably significant for defect classification and they listed coupling metrics according to their rank.

Faruk et al. [24] proposed a new classification model for software defect prediction in their paper. They used artificial neural networks and novel artificial bee colony algorithm to classify software faults. They used five NASA datasets to perform their evaluations containing different code metrics and used several classification related performance measures like accuracy and area under the curve.

They concluded that their classification performed well and could be used for fault prediction purposes.

2.2 Fault Prediction by Process Metrics

Process metrics are investigated in fewer studies as compared to code metrics. Many researchers claim process metrics to be more informative and better predictors than code metrics. Following are some papers that are related to our research which used process metrics in their studies.

Matsumoto et al. [25] investigated the significance of developer metrics. They used fifteen code and seven change metrics to propose and evaluate developer metrics. The Regression and Classification Tree was used to evaluate the relation between defects in the code and various developer metrics. They found that the incidence of defects differs from developer to developer and that the modules edited by different developers can have larger number of defects.

Marian et al. [26] investigated the significance of process metrics for software fault prediction. They constructed and compared two models based on code and four change metrics which were number of defects in previous versions, number of distinct committers, number of revisions and number of modified lines. First model was built using code metrics only and in second model they added on change metric with code metrics and repeated separately for all four change metrics. They used Ckjm tool for extracting code metrics and BugInfo tool for change metrics and number of faults. They performed their evaluations using forty-three public and twenty-seven industrial datasets. They applied stepwise linear regression as machine learning model. Their results showed that number of modified lines and number of distinct committers are important metrics for fault prediction and change metrics are equally informative as code metrics.

Kumar et al. [27] have applied Decision Tree Regression for estimating the number of defects. They performed a comparison between inter and intra release defect

prediction. The models were built using nineteen PROMISE data sets. Their results implied that intra-release fault prediction has performed better as compared to inter-release fault prediction.

Florence et al. [28] employed genetic algorithms to refine features and they used deep neural networks for fault classification.

They used PROMISE datasets to evaluate their models and accuracy measure as performance metric. They summarized that their proposed method presented better results than other classifiers.

2.3 Feature Selection for SFP

Many researchers have applied different feature selection techniques to improve fault prediction process. Some researchers have also performed comparisons of different feature selection methods used in software fault prediction. We have summarized and listed the following research papers which involved feature selection.

Laradgi et al. [29] introduced an algorithm for classification of software defects having two types. They combined feature selection with ensemble learning to perform the classification. The first type of algorithm used feature selection with ensemble learning for classification and the second type deals only with ensemble learning (without feature selection).

They evaluated Greedy Forward Selection and Pearson's Correlation and found that greedy selection produced better results than Pearson's Correlation. They performed their experiments using multiple NASA datasets having code metrics. They concluded that their proposed algorithm with feature selection and ensemble approach performed remarkably for classification of software defects.

Wang et al. [30] performed a comparison of different feature ranking techniques for classification of software defects. They used ensemble learning on 16 public datasets from eclipse, NASA and telecommunication software system having

multiple code metrics. They used different filter base ranking techniques in combination with naïve bayes classifier to build and evaluate seventeen ensembles. They concluded that individual ensembles do not outperform others but few ensembles have equally better performance as compared to remaining models.

Khoshgoftaar et al. [31] compared four combinations of feature selection and modeling techniques for software defect classification. They used feature selection and data sampling to improve the performance of their classifiers. They performed their experiments using K-Nearest Neighbors and Support Vector Machine on nine PROMISE datasets and used Area Under the Curve as their performance evaluator. They concluded that sampling has better effect on feature selection and does not play important role in defect prediction.

In another paper [32], khoshgoftaar et al. investigated the impact of data sampling and attribute selection techniques for software defect classification. They compared five variations of data having sampling/non sampling and attribute selection/no attribute selection as variation criteria. They used eight public datasets from NASA to perform their evaluations. They concluded that class balancing plays an important role in improving classification prediction performance with attribute selection having very little improvement as compared to balancing.

Wahono et al. [33] investigated the effect of particle swarm optimization on software fault prediction. They used particle swarm optimization for feature selection and combined it with bagging to perform their evaluations. They used eleven classifiers in their bagging ensemble and constructed models using nine NASA datasets having code metrics. They concluded that their classification models outperformed significantly.

Wang et al. [34] combined attribute ranking with ensemble learning for classification of software faults. They compared the performance of six filter-based ranking techniques using their ensemble. They used three NASA datasets to perform evaluations and used area under the curve as their performance measure. They concluded that ensemble methods performed better for fault prediction and different ranking techniques have different impact on the prediction process.

Catal et al. [35] evaluated the impact of dataset size, feature set and attribute selection methods for software fault prediction. They constructed nine classification models using five public NASA datasets having different code metrics. They used area under the curve and accuracy as their performance measure. According to them, naïve bayes performed better when used for small datasets and random forest is best suited for large size datasets.

Chao et al. [36] proposed a novel feature selection method called feature selection using clusters of hybrid data (FeSCH). Their method used density of clusters and ranking of features for feature selection. They used their selection technique to evaluate cross project defect prediction. They used AEEEM and Relink CPDP datasets having different type of code metrics and used F-measure as their performance measure. Their results showed that FeSCH performed better as compared to other feature selection methods and its performance was independent of the classifier used.

Bayesian network has been used by Okutan and Yildiz [37] to select essential Software metrics for classifying defect-prone program modules utilizing object-oriented approaches. They used nine PROMISE registry datasets to perform evaluations. Their study also identified two new metrics that are number of developers and lack of quality coding. They stated that RFC (Response for Class), LOC (Lines of Code) and LCQ (Lack of Coding Quality) are very important metrics for defect prediction.

Through the application of filter based feature selection for defect proneness classification, a smaller subset of code metrics was selected by He et al. [38]. They further reduced metrics based on top-k and redundancy criteria. Thirty-four PROMISE databases having code metrics were used in their study. They evaluated their selected feature set using six machine learning classifiers and presented the results through recall, F-measure and precision.

Maqsood et al. [39] evaluated the importance of nine classification models of machine learning. They used SMOTE and re-sampling to balance the datasets and selected important metrics based on Fisher linear discrimination analysis.

They chose top four classifiers and evaluated fifteen PROMISE datasets which were composed of code metrics. They presented results by recall, precision and F-measure performance metrics.

Turabieh et al. [40] used layered recurrent neural networks in a proposed attribute selection method to decrease the number of code metrics for software defect classification. Nineteen PROMISE datasets were used for evaluation and results were presented using AUC. They concluded that their proposed method performed much better when compared to five other ML techniques. A comparison of all the papers mentioned in this chapter is given in the following table:

TABLE 2.1: Literature Analysis

Sr #	**Paper**	**Metrics**	**SFP Method**	**ML Algo/model**	**Dataset**
1.	Tibor Gyimothy [12]	Code	Regression	Linear Regression, Decision Tree, Neural Network	Mozilla, Bugzilla
2.	Zimmermann et al. [13]	Code	Regression	Linear Regression	Eclipse 2.0,2.1,3.0
3.	Khoshgoftaar et al. [14]	Code	Regression	Zero-inflated, hurdle negative binomial regression	Industrial
4.	Zhang et al. [15]	Code	Classification	Naïve Bayes, Logistic Regression, Neural Networks	NASA
5.	Marian et al. [16]	Code	Regression	Stepwise Linear Regression	NASA

Sr #	Paper	Metrics	SFP Method	ML Algo/model	Dataset
6.	Erturk et al. [17]	Code	Classification	Fuzzy Inference	PROMISE
7.	Fagundes et al. [18]	Code	Regression	Zero-inflated Poisson Regression, and Hurdle Regression	NASA
8.	Rathore et al. [19]	Code	Regression	Linear Regression, Multi Layer Perceptron, Naïve Bayes Regression	PROMISE
9.	Shatnawi [20]	Code	Classification	ROC Analysis	N/A
10.	Rathore et al. [21]	Code	Regression	Linear Regression, Decision Tree Regression, Genetic Programming	PROMISE
11.	Rashid et al. [22]	Code	Classification	Artificial Neural Network	N/A
12.	Rizwan et al. [23]	Code	Classification	Support Vector Machine	N/A
13.	Faruk et al. [24]	Code	Classification	Artificial Neural Network	NASA

Sr #	Paper	Metrics	SFP Method	ML Algo/model	Dataset
14.	Matsumoto et al. [25]	Process	Regression	CART	N/A
15.	Marian et al. [26]	Process	Regression	Stepwise Linear Regression	N/A
16.	Kumar et al. [27]	Process	Regression	Decision Tree Regression	PROMISE
17.	Florence et al. [28]	Process	Classification	Genetic Algorithms, Neural Networks	PROMISE
18.	Laradgi et al. [29]	Code	Classification	Greedy Forward Selection, Pearson Correlation	NASA
19.	Wang et al. [30]	Code	Classification	Naïve Bayes	NASA, Eclipse
20.	Khoshgoftaar et al. [31]	Code	Classification	KNN, Support Vector Machine	PROMISE
21.	khoshgoftaar et al.[32]	Code	Classification	N/A	NASA
22.	Wahono et al. [33]	Code	Classification	Particle Swarm Optimization	NASA

Sr #	Paper	Metrics	SFP Method	ML Algo/model	Dataset
23.	Wang et al. [34]	Code	Classification	Ensemble Learning	NASA
24.	Catal et al. [35]	Code	Classification	Naïve Bayes	NASA
25.	Chao et al. [36]	Code	Classification	FS using Clusters of Hybrid Data	AEEEM, Relink
26.	Okutan and Yildiz [37]	Code	Classification	Bayesian Network	PROMISE
27.	He et al. [38]	Code	Classification	top-k features	PROMISE
28.	Maqsood et al. [39]	Code	Classification	Fischer Linear Discriminant Analysis	PROMISE
29.	Turabieh et al. [40]	Code	Classification	Layered Recurrent Neural Networks	PROMISE

After evaluation of state-of-the art approaches we concluded that although software fault prediction is thoroughly addressed in research and many researchers have performed studies to evaluate different type of fault prediction methods, there are some areas of Software Fault Prediction which are not addressed much and need more investigation. The first area is the use of process metrics which were not used as much as code metrics in studies. Second area is the evaluation of regression techniques in Software Fault Prediction for number of faults prediction

which also need more investigation. The third area is the combined use of code and process type of metrics and applying feature selection to improve prediction process using combined selection of both type of metrics. Therefore, we have aimed our research on all these areas.

Chapter 3

Proposed Approach

In this chapter we discuss about the methodology and techniques used in our study. Here we explain the research method used in our study, features and types of features used in our experiments, description of datasets used, machine learning models and performance metrics used in our research.

3.1 Research Methodology

To study the significance of process metrics for predicting the number of faults in a software system and to identify the most important and influential software metrics (both process and code metrics) that affect the fault prediction process, we formed an approach in which two steps are performed.

In the first step, we merged instances from all five datasets into a single dataset to increase number of instances and then used wrapper subset selection technique to select best feature subsets from process and code metrics respectively. The output of feature selection method results in subset of seven out of fifteen process metrics and subset of eleven out of seventeen code metrics.

In the second step, we added both types of selected metric subsets to make hybrid set of metrics which was then used for prediction of number of faults. We also

divided selected process and code metrics into categories and evaluated models for categories as well as performance of single metrics from selected process and code subsets. The results of these experiments are described in detail in results and discussions chapter.

A diagram depicting the overall steps of our proposed framework (feature selection and prediction process) is presented in Figure 3.1.

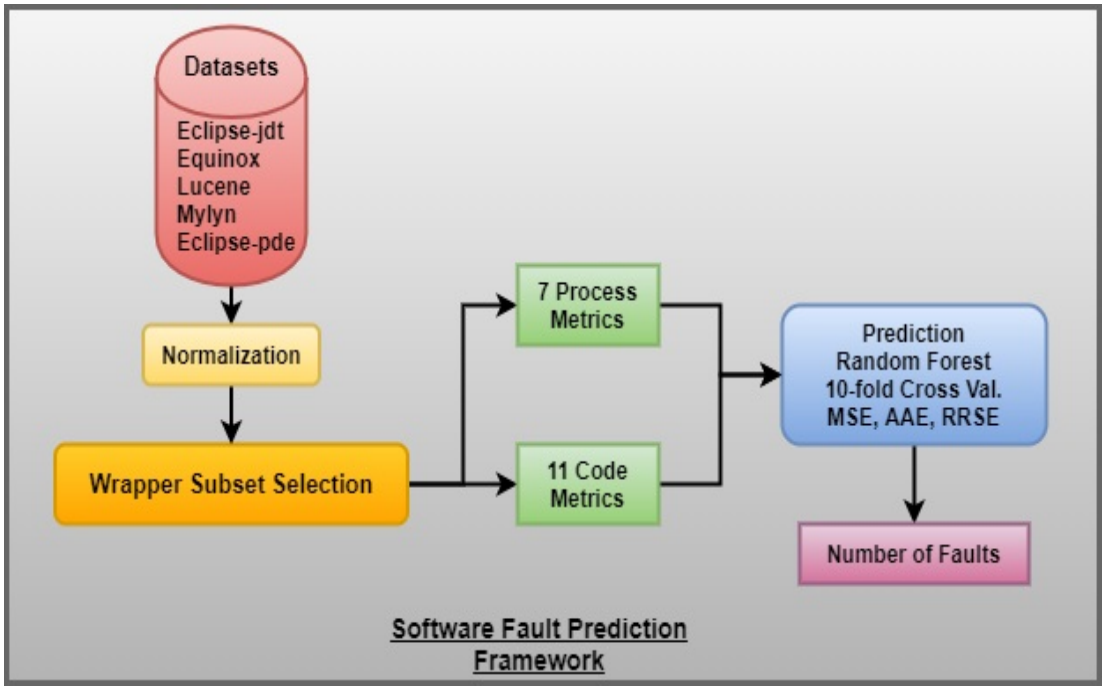

FIGURE 3.1: Software Fault Prediction Framework

3.2 Features

In this section, we will give brief description of features that are used in our research. The features which we used in our research are two type of software metrics namely code and process metrics. Each of these types contain multiple metrics. The software metrics calculated by using software's coding structure are called code metrics. They consist of information of code properties, size of code and structure of code. The other type of software metrics used in our research are process metrics which are based on edits, revisions and changes to a software. Below is detailed description of process and code metrics used in our study.

3.2.1 Process Metrics

As discussed above, process metrics are calculated using change information of a software. In our research, fifteen different process metrics are used for experimentation. These metrics contain information about added or deleted lines of code, code churn, number of authors and number of versions etc. Following is the list of these fifteen process metrics having name and description:

1. **NumberOfVersions:** Number of versions produced since the first release.
2. **NumberOfFixes:** Number of times a module is treated for bug fixing.
3. **NumberOfRefactorings:** Number of times a module has been refactored. Refactoring refers to change in code design without changing the output of the software.
4. **NumberOfAuthors:** Number of authors who made commits to the software module
5. **Age:** Age of a module since its first release.
6. **WeightedAge:** Age of a module normalized by added lines of code.
7. **LinesAdded:** Total number of lines added since last version
8. **MaxLinesAdded:** Max lines added in all commits.
9. **AvgLinesAdded:** Average lines added in all commits.
10. **LinesRemoved:** Total number of lines deleted since last version
11. **MaxLinesRemoved:** Max lines deleted in all commits.
12. **AvgLinesRemoved:** Average lines deleted in all commits.
13. **CodeChurn:** Code churn is the change in code; addition or deletion, over a specific period of time.
14. **AvgCodeChurn:** Average code churn per commit.
15. **MaxCodeChurn:** Max code churn per commit.

3.2.2 Code Metrics

The second type of metrics used in our study is code metrics. In this study, we have used seventeen different code metrics for experimentation purposes. These code metrics consists of lines of code (size), coupling between objects, cohesion among modules, hierarchy of classes and other object-oriented metrics.

We have listed all seventeen code metrics containing the name and brief description of each metric as follows:

1. **DIT (Depth of Inheritance Tree):** Depth of a class within the class hierarchy from the root of inheritance.
2. **FanIn:** Number of functions/methods that call a given function.
3. **FanOut:** Numbers of functions/methods that are called by a given function.
4. **LCOM (Lack of Cohesion Methods):** Count of the methods that do not share variables and fields with other methods.
5. **NOC (Number of Children):**Number of child classes of a class.
6. **NumberOfAttributes:** Total number of attributes of a class.
7. **NumberOfAttributesInherited:** Total number of inherited attributes of a class.
8. **NumberOfLinesOfCode:** Total lines of codes in a class/module.
9. **NumberOfMethodsInherited:** Total number of methods inherited by a class.
10. **NumberOfPrivateMethods:** Total number of the private methods in a class.
11. **NumberOfPublicAttributes:** Total number of public attributes in a class.

12. **CBO (Coupling between objects):** Measure of modules that can access a given module and other modules that this module can access.

13. **NumberOfMethods:** Number of methods or functions in a class.

14. **NumberOfPrivateAttributes:** Measure of private attributes in a class.

15. **NumberOfPublicMethods:** Measure of public methods in a class.

16. **RFC (Response for Class):** Measure of methods that can be accessed by objects of given class.

17. **WMC (Weighted Methods per Class):** Measure of complexities of all methods in a class.

3.3 Experimental Setup

In experimental setup section we discuss the details of datasets and machine learning model used in our research. The five datasets used here are listed in below (dataset) section containing number of instances for each dataset as well as their respective description in a table. Further, we have discussed the machine learning model and its configuration used for experiments.

3.3.1 Datasets

In our study we have used five datasets containing count of faults/bugs. The datasets are publicly available and have count of faults as resultant class. These datasets belong to open source eclipse projects and are presented by [41]. Names of these datasets are eclipse-jdt, equinox, lucene, mylyn and eclipse-pde. All five datasets contain process and code metrics as features. There are fifteen process metrics and seventeen code metrics for each dataset. The information about all these metrics are discussed in details in features section. These datasets are class level datasets and each of them has number of faults/bugs as the target class. The

number of instances for datasets ranges from 324 to 1862 and more details are given in the following table (Table 3.1).

TABLE 3.1: Datasets Information

Dataset	**Description**	**Instances**
Eclipse-jdt	Java infrastructure of the Eclipse Java IDE	997
Equinox	OSGi framework implementation used for all of Eclipse	324
Lucene	Open-source Apache search software	691
Mylyn	Task and application lifecycle management framework for Eclipse	1862
Eclipse-pde	User interface that provides a set of tools to create and develop Eclipse plug-ins and features	1497

3.3.2 Machine Learning Model

In this section, we explain the machine learning model used for the experimentation in our research. Due to its wide acceptance and proven accuracy, we used the Random Forest as machine learning model for training and testing in our study [42].

Random forest was introduced in 1995 and developed as a machine learning technique in 2001. It can be used for both classification and regression learning. Structure wise random forest is an ensemble method which consists of multiple decision trees that correspond to separate classes. The output is mode of all trees for classification and mean of all trees is computed for regression learning.

A basic structural design of random forest is given in the following picture (Figure 3.2):

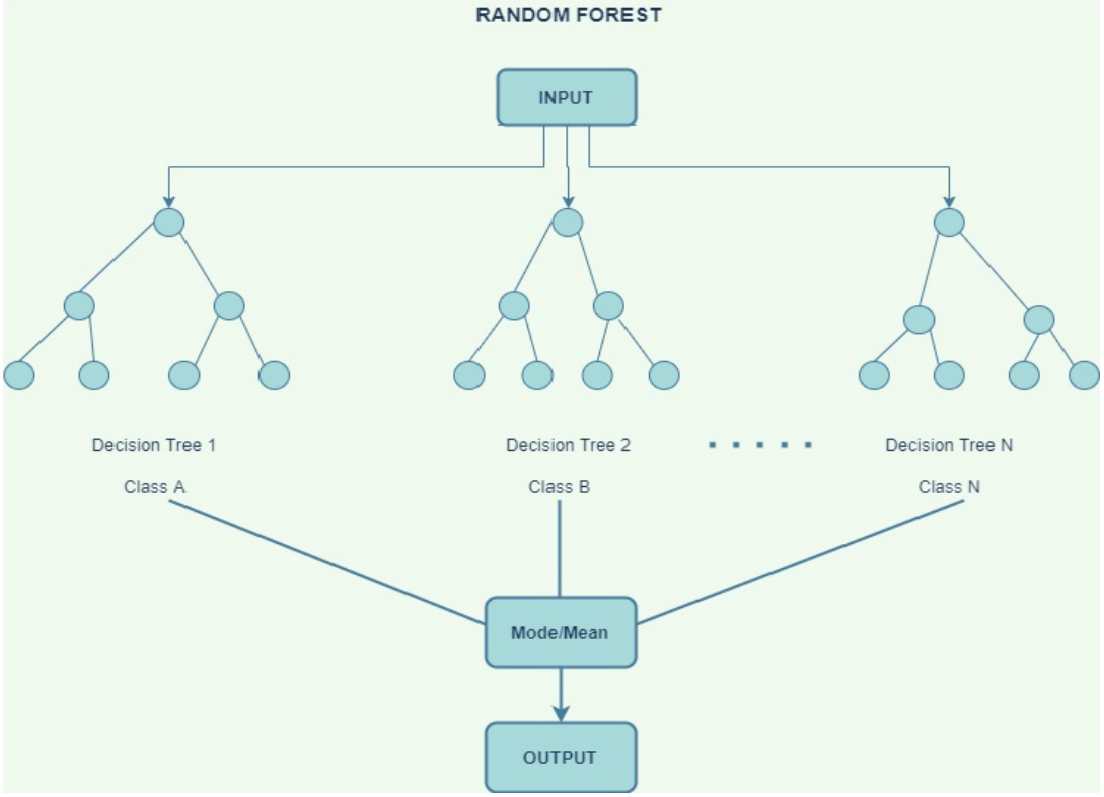

FIGURE 3.2: Random Forest Basic Diagram

3.4 Evaluation Metrics

The performance metrics used in this study are well known metrics to evaluate the regression results as suggested by prior studies [14]. We have used three performance metrics in our study. These are Mean Squared Error (MSE) [43], Root Relative Squared Error (RRSE) and Average Absolute Error (AAE) [44].

3.4.1 Mean Squared Error (MSE)

The mean squared error is widely used as a performance measure in regression problems. Its formula is given as

$$\text{MSE} = \frac{1}{\text{k}} \sum_{\text{i}=1}^{\text{k}} (\text{Y}_\text{i} - \text{Y}'_\text{i})^2 \tag{3.1}$$

Where Y_i is the vector of observed values and Y'_i is for predicted values.

3.4.2 Root Relative Squared Error (RRSE)

The formula for root relative squared error is given as follows:

$$\mathrm{RRSE} = \sqrt{\frac{\sum_{\mathrm{k=1}}^{\mathrm{n}} \left(\mathrm{P_{ik}} - \mathrm{T_k}\right)^2}{\sum_{\mathrm{k=1}}^{\mathrm{n}} \left(\mathrm{T_k} - \overline{\mathrm{T}}\right)^2}} \tag{3.2}$$

where P_{ik} is predicted value of model i for record k out of n, T_k is target value and $\overline{T} = \frac{1}{\mathrm{n}} \sum_{\mathrm{k=1}}^{\mathrm{n}} \mathrm{T_k}$

3.4.3 Average Absolute Error (AAE)

Average absolute error is also used commonly as performance measure in regression techniques. It was introduced by [44]. The formula for average absolute error is given as:

$$\mathrm{AAE} = \frac{1}{\mathrm{k}} \sum_{\mathrm{i=1}}^{\mathrm{k}} \left| \left(\mathrm{Y_i'} - \mathrm{Y_i}\right) \right| \tag{3.3}$$

Where Y_i is the vector of observed values and Y_i' is for predicted values.

Chapter 4

Results and Discussion

In the previous chapter, we have explained the in-depth details of the proposed methodology including feature sets, datasets used, machine learning model and performance metrics used. This chapter focuses on the results that have been obtained by applying the proposed methodology. Moreover, this chapter is divided into six parts having results from feature selection, performance of selected hybrid metrics set, comparison and evaluation of types of selected metrics, their categories, individual selected metrics evaluation and in the last part discussion and implications.

4.1 Feature Selection

In this part, we discuss the feature selection process used in our research in details. We have used Wrapper subset selection method for feature selection and random forest as the induction algorithm for wrapper selection.

Wrapper subset selection was introduced by (Kohavi and John 1997) in 1997 [45]. It is an exhaustive feature selection method which makes all possible feature subset and use an induction algorithm as its evaluation criteria. Then it evaluates all subsets of features against the performance measure of induction algorithm and selects subsets with best results.

The performance metric of the induction algorithm serves as the selection criteria in wrapper subset selection. The advantage of wrapper subset selection over other feature selection methods is that it gives improved performance if the count of features is large due to its exhaustive search approach but at the cost of computation power and time.

In our research, the wrapper subset selection method is applied for identification of most important feature sets from process metrics and code metrics. We have performed this selection on a combined dataset which was composed of instances from all (five) datasets. The selected process and code metrics are further used for evaluation and comparisons.

The selected process metrics are listed in Table 4.1 and selected code metrics in Table 4.2.

We have also combined selected process and code metrics into hybrid metrics set.

Table 4.1: Selected Process Metrics

Sr. #	**Type**	**Metric**
1.	Change based	Number Of Versions
2.	Change based	Number Of Fixes
3.	Change based	Number Of Refactoring
4.	Change based	AvgCodeChurn
5.	Author based	NumberOfAuthors
6.	Age based	Age
7.	Age based	Weighted Age

TABLE 4.2: Selected Code Metrics

Sr. #	**Type**	**Metric**
1.	Inheritance based	DIT
2.	Inheritance based	NOC
3.	Inheritance based	Number Of Attributes Inherited
4.	Inheritance based	Number Of Methods Inherited
5.	Method call based	Fan In
6.	Method call based	Fan Out
7.	Cohesion based	LCOM
8.	Structure based	Number Of Attributes
9.	Structure based	Number Of Lines Of Code
10.	Access based	Number Of Private Methods
11.	Access based	Number Of Public Attributes

To continue, we arranged the selected code and process metrics category wise depending on the nature of metrics, as seen in Table 4.1 and Table 4.2. The seven selected process metrics were divided into three categories namely change based, author-based, and age-based. The eleven selected code metrics were divided into five categories which are method call based, inheritance based, structure based, cohesion based, and access based.

The composition of all categories of process and code metrics is given in Table 4.1 and Table 4.2. The comparison and significance of these categories is given in section 4.4 separately for process and code metric categories.

4.2 Predictive Performance Analysis of Hybrid Metrics

In this part, we evaluated the performance of hybrid subset of process and code metrics with all process and code metrics set. Random Forest was used as a machine learning model for experimentation with 10-fold cross validation. The evaluation was performed using MSE, AAE and RRSE performance metrics. Seven process and eleven code metrics were selected in feature selection process using wrapper subset selection which were joined as eighteen selected hybrid metrics. For comparison all fifteen process and all seventeen code metrics were joined as all hybrid metrics. Results make it evident that the performance of selected hybrid set is much better than that of all hybrid metrics which can be seen in Table 4.3. Results of all five datasets arranged according to three performance measures for selected hybrid and all hybrid set is given in the following table (Table 4.3):

Table 4.3: Performance Analysis using Hybrid Metrics

Datasets	**All Hybrid Metrics**			**Selected Hybrid Metrics**		
	MSE	**AAE**	**RRSE(%)**	**MSE**	**AAE**	**RRSE(%)**
Eclipse-jdt	0.6331	0.3939	76.3836	**0.6147**	**0.3863**	**75.2634**
Equinox	0.8947	0.5531	64.8926	**0.8825**	**0.5459**	**64.4451**
Lucene	0.2289	0.1884	80.4432	**0.2122**	**0.1805**	**77.4648**
Mylyn	0.3097	0.2608	92.8568	0.3163	**0.2545**	93.8518
Eclipse-pde	0.7974	0.3269	93.3838	**0.7721**	**0.3225**	**91.88**

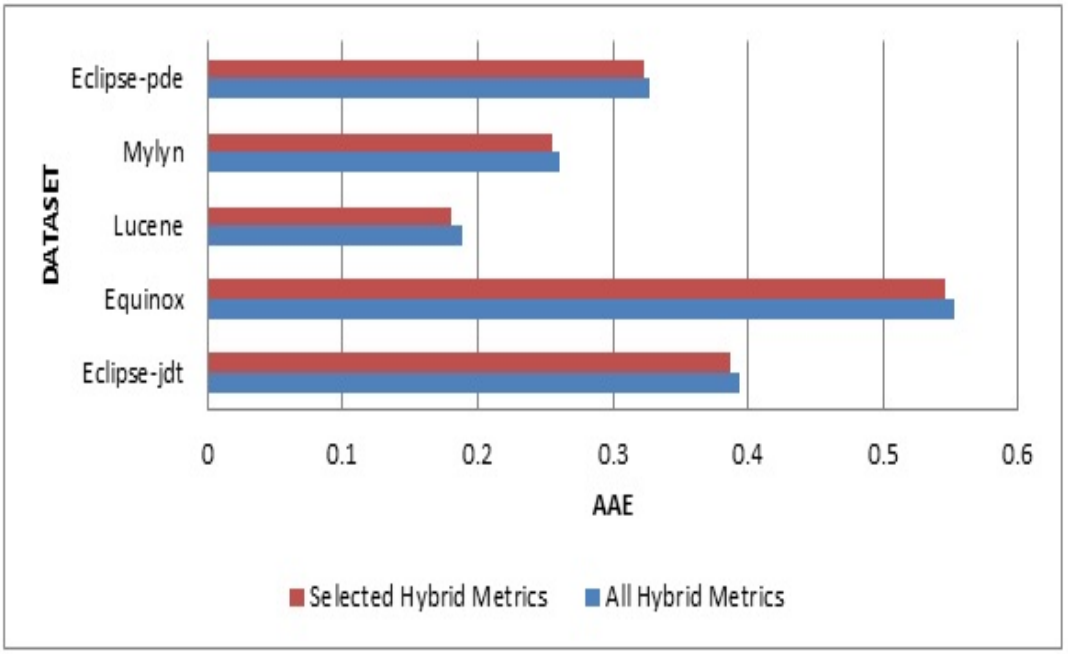

FIGURE 4.1: Hybrid Metrics Performance

It can be seen from Figure 4.1 that the selected hybrid metrics set performed better than all hybrid metrics set for all five datasets.

This results of all the three performance measures were improved in four datasets and for Mylyn improvement was seen for Average Absolute Error (AAE). The best improved performance was seen in eclipse-pde in which Mean Squared Error (MSE) had a difference of 0.0253, Average Absolute Error (AAE) having difference of 0.0044 and in Root Relative Squared Error (RRSE) difference was 1.5038%.

Following eclipse-pde, eclipse-jdt produced next best results using selected hybrid metrics with a reduction of 0.0184 in Mean Squared Error (MSE), 0.0076 in AAE and 1.1202% in Root Relative Squared Error (RRSE). In case of Mylyn, improved performance was only seen for Average Absolute Error (AAE) having a reduction of 0.0063.

These results not only prove the importance and effectiveness of feature selection process for software metrics but also show the significance of selected process and code metrics which performed better as compared to all process and code metrics used in this study.

4.3 Feature Type wise Analysis

In this portion, we present the results of selected process and code metrics compared to all process and code metrics respectively. The selected process metrics performed better in comparison with all process metrics which can be seen in Table 4.4.

The configuration of experiments is same as before with random forest as machine learning model and 10-fold cross validation. The results were presented using Mean Squared Error (MSE), Average Absolute Error (AAE) and Root Relative Squared Error (RRSE) as evaluation metrics.

The results show that performance was improved for all five datasets when using selected process metrics set. The most improvement in results was observed for equinox, eclipse-jdt and lucene where highest difference was seen in equinox's Mean Squared Error (MSE) of 0.0265 followed by eclipse-jdt having difference of 0.0200 in Mean Squared Error (MSE). For Mylyn the reduction in Average Absolute Error (AAE) was observed having difference of 0.0007.

We have also presented these results using AAE measure in Figure 4.2.

Table 4.4: Performance Analysis using Process Metrics

Datasets	**All Process Metrics**			**Selected Process Metrics**		
	MSE	**AAE**	**RRSE(%)**	**MSE**	**AAE**	**RRSE(%)**
Eclipse-jdt	0.6727	0.4129	78.7316	**0.6527**	**0.4032**	**77.5594**
Equinox	1.0084	0.5697	68.8938	**0.9819**	**0.5569**	**67.9795**
Lucene	0.2161	0.1713	78.1754	**0.1977**	**0.1623**	**74.7697**
Mylyn	0.3146	0.2616	93.5918	0.3342	**0.2609**	96.4592
Eclipse-pde	0.8317	0.3412	95.3763	**0.8245**	**0.3294**	**94.9531**

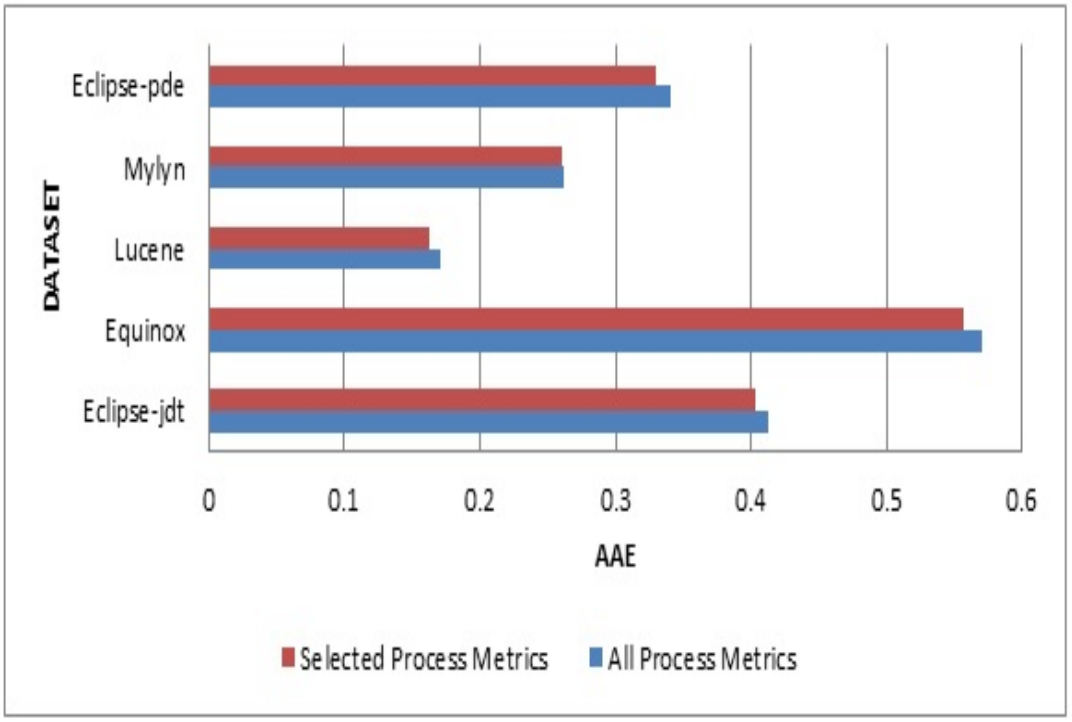

FIGURE 4.2: Selected Process Metrics Performance

Continuing the evaluation of selected metric types, we also compared the selected code metrics set (eleven metrics) with all code metrics used in our study which count to seventeen. The configurations of experiments were same as for selected process metrics.

The selected code metrics also showed the better results as compared to all code metrics along all the five datasets. Most improvement was observed for the equinox, eclipse-jdt and lucene which had improved results in all three performance metrics.

The most improvement was calculated for equinox having a difference of 0.0472 in Mean Squared Error (MSE). The second best results were shown by eclipse-jdt having difference of 0.0142 in Mean Squared Error (MSE). In case of eclipse-pde and Mylyn, the performance also improved with Mylyn having Average Absolute Error (AAE) difference of 0.0017 and eclipse-pde having Average Absolute Error (AAE) difference of 0.0005.

The results are given in the following table (Table 4.5).

TABLE 4.5: Performance Analysis using Code Metrics

Datasets	**All Code Metrics**			**Selected Code Metrics**		
	MSE	**AAE**	**RRSE(%)**	**MSE**	**AAE**	**RRSE(%)**
Eclipse-jdt	0.6587	0.3915	77.9088	**0.6445**	**0.3865**	**77.0622**
Equinox	1.1004	0.604	71.9684	**1.0531**	**0.5956**	**70.4031**
Lucene	0.3120	0.2234	93.926	**0.3072**	**0.2188**	**93.2164**
Mylyn	0.3221	0.2699	94.688	0.3274	**0.2682**	95.4731
Eclipse-pde	0.8473	0.33	96.2691	0.8493	**0.3295**	96.3777

Following figure (Figure 4.3) contains the graphical representation of performance of code metrics using AAE measure.

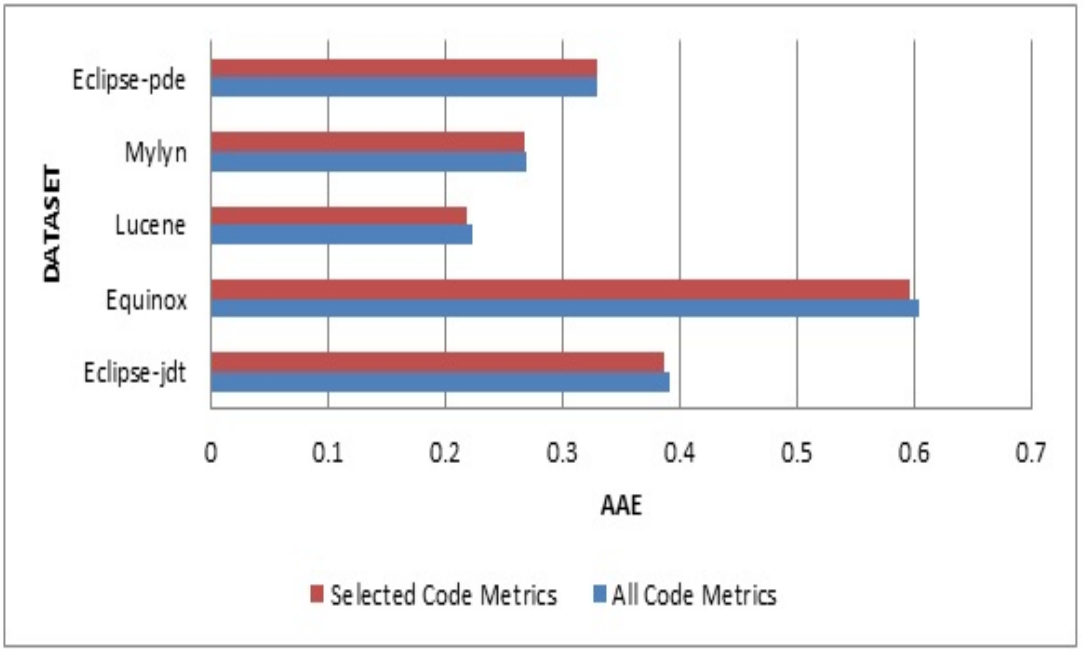

FIGURE 4.3: Selected Code Metrics Performance

These results of selected metric types also show that overall process metrics showed better results than code metrics and selected metrics sets of each type (process and code) performed better than their corresponding all metrics sets.

4.4 Category wise Analysis

In the previous section we compared the performance of selected process and code metrics subsets which were obtained using feature selection. These selected features were combined into categories according to their nature and type which are given in Table 4.1 and Table 4.2 in Feature selection section 4.1. We also performed comparisons among categories of each (process and code) type of metrics separately. These results are explained in the following two sections first for process metric categories and then for code metric categories.

4.4.1 Process Metrics Categories

The selected process metrics were divided into three categories which were Age based, Author based and Change based. The comparison among these categories were performed using random forest on all five datasets by selecting features of one category at a time and results were compiled using MSE measure. The results show that the best results were obtained for Change based category having least MSE overall followed by Author based and Age based categories. The performance of process metrics categories is given in Figure 4.4 and complete results are given in Table 4.6.

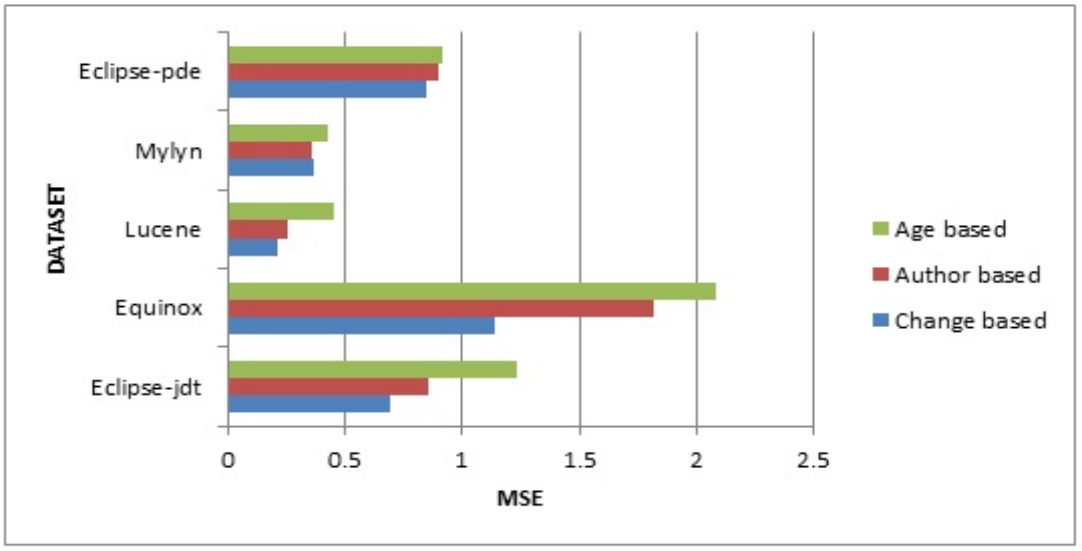

FIGURE 4.4: Process Metrics Category wise Significance

TABLE 4.6: Process Metrics Category Results

Datasets	**Age based**			**Author based**			**Change based**		
	MSE	**AAE**	**RRSE**	**MSE**	**AAE**	**RRSE**	**MSE**	**AAE**	**RRSE**
Eclipse-jdt	1.2274	0.5467	106.35	0.8636	0.4787	89.20	0.6936	0.421	79.94
Equinox	2.0811	0.7537	98.96	1.8193	0.794	92.53	1.1383	0.6243	73.19
Lucene	0.4541	0.2273	89.02	0.2579	0.2008	85.38	0.2125	0.1678	77.51
Mylyn	0.4275	0.2895	109.08	0.3576	0.3141	99.77	0.3739	0.2919	102.03
Eclipse-pde	0.9172	0.3611	100.15	0.9048	0.3746	99.47	0.8512	0.3419	96.48

4.4.2 Code Metrics Categories

For evaluating selected code metrics categories which were five in number, no category presented consistent performance for all five datasets. The Method Call based, Cohesion based and Access based categories showed a minor reduction in MSE in three out of five datasets as compared to other categories. It can be said that all of the categories of selected code metrics are equally important in their predictive ability. The comparison of selected code metrics categories can be seen in Figure 4.5 and complete results are also given in Table 4.7.

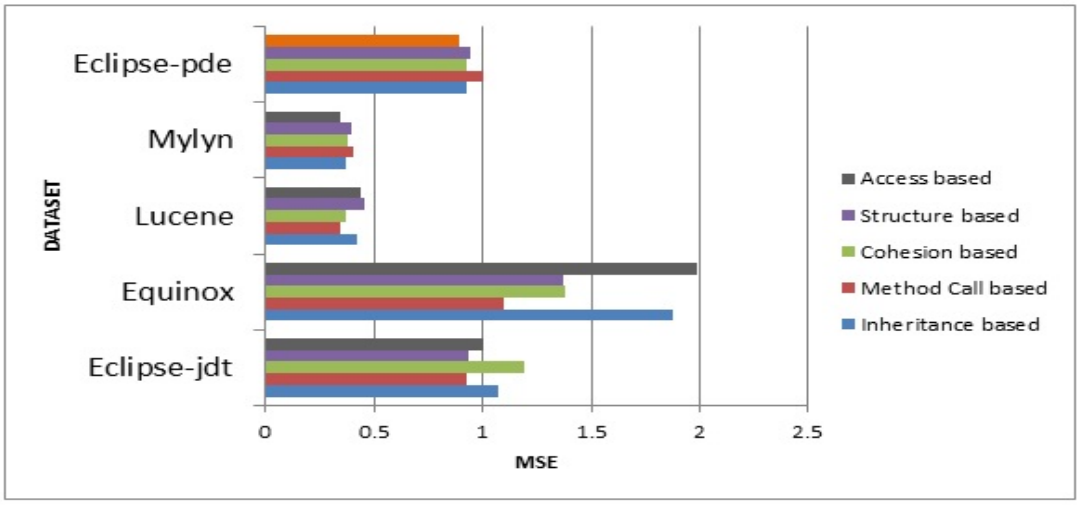

FIGURE 4.5: Code Metrics Category wise Significance

Table 4.7: Code Metrics Category Results

Dataset	Access based			Structure based			Cohesion based			Method Call based			Inheritance based		
	Mean Square Error (MSE)	Avg Absolute Error (AAE)	Root Relative Square Error (RRSE)	Mean Square Error (MSE)	Avg Absolute Error (AAE)	Root Relative Square Error (RRSE)	Mean Square Error (MSE)	Avg Absolute Error (AAE)	Root Relative Square Error (RRSE)	Mean Square Error (MSE)	Avg Absolute Error (AAE)	Root Relative Square Error (RRSE)	Mean Square Error (MSE)	Avg Absolute Error (AAE)	Root Relative Square Error (RRSE)
Eclipse-jdt	1.0052	0.5207	96.25	0.9357	0.4595	92.86	1.1885	0.539	104.65	0.9262	0.4633	92.38	0.9262	0.4633	92.38
Equinox	1.9912	0.8505	96.81	1.3675	0.6376	80.22	1.3799	0.7155	80.59	1.1008	0.65	71.98	1.1008	0.65	71.98
Lucene	0.4413	0.2479	111.71	0.4582	0.2557	113.83	0.3653	0.238	101.64	0.3470	0.2298	99.07	0.3470	0.2298	99.07
Mylyn	0.3425	0.281	97.64	0.3920	0.2823	104.46	0.3808	0.3059	102.97	0.3997	0.2856	105.49	0.3997	0.2856	105.49
Eclipse-pde	0.8955	0.3542	98.97	0.9397	0.3452	101.38	0.9297	0.3722	100.83	1.0046	0.3782	104.82	1.0046	0.3782	104.82

4.5 Individual Feature Evaluation

After performing comparisons of types of selected metrics and their categories, we also evaluated the effectiveness and importance of each metric among selected process and code metrics. Again, the configurations were same as before with random forest as machine learning model and 10-fold cross validation. We performed experiments by individually selecting selected metrics for model building and arranged the results according to metrics type (process and code) using MSE. Results of individual process metrics can be seen in Figure 4.6. Among the seven selected process metrics, number Of Authors performed best for all five datasets followed by number Of Versions having reduced MSE for four datasets. The age With Respect To, number Of Fixes and number Of Refactorings showed better results for three datsets whereas weightedAge With Respect To and avgCodeChurn had highest MSE in all five datasets.

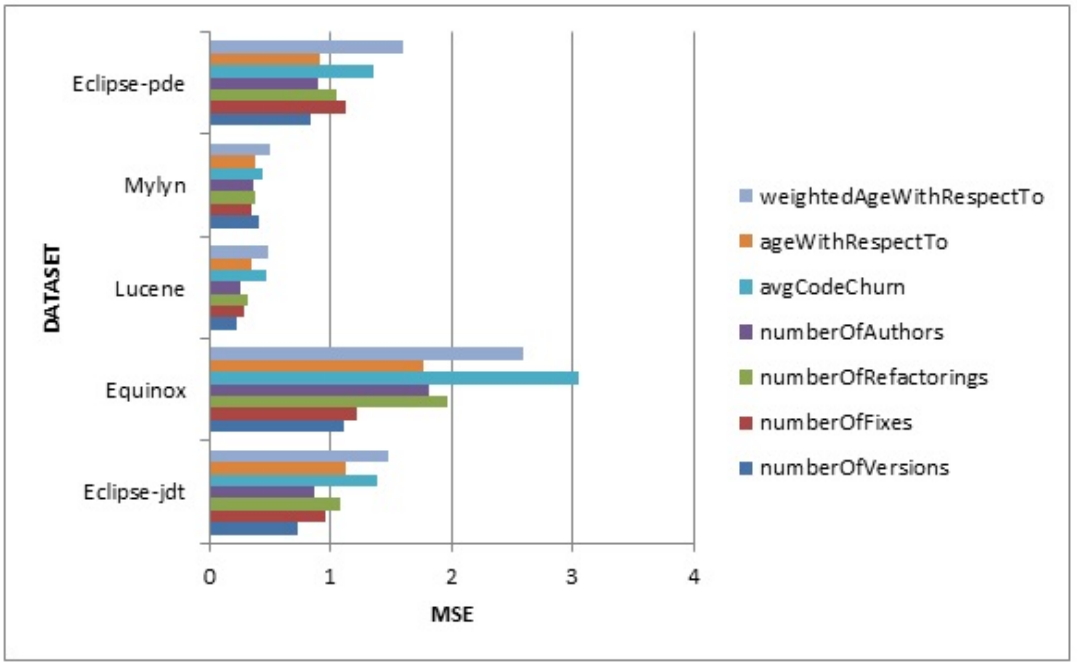

FIGURE 4.6: Selected Process Metrics Significance

When comparing the selected code metrics, similar variable pattern of performance was observed as was seen in process metrics. Among eleven code metrics, none performed better for all five datasets. The best results were calculated for FanOut which showed reduced MSE in four datasets.

Six metrics; numberOfAttributesInherited, DIT, numberOfPrivateMethods, NOC, numberOfPublicAttributes and numberOfLinesOfCode performed better in three datasets. Three code metrics namely numberOfMethodsInherited, lcom and fanIn performed better in two out of five datasets whereas, numberOfAttributes had lower MSE in one dataset only. The results of individual selected code metrics can be seen in the following figure (Figure 4.7):

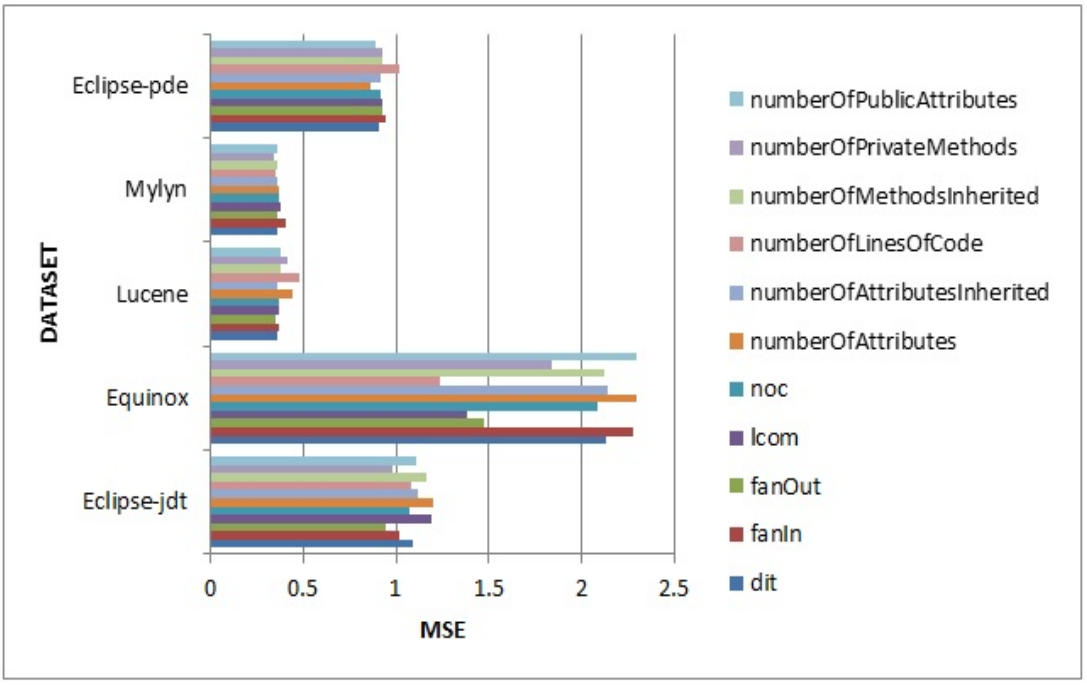

FIGURE 4.7: Selected Code Metrics Significance

4.6 Comparison with Existing Work

For validation of our results, we have compared our hybrid selection results with the results presented in [19]. We have chosen this paper for our results validation because it uses the same regression datasets as used in our paper and performed ensemble machine learning modeling to predict the number of faults in a software system. It is to be noted that there are very few studies that used the same eclipse datasets and majority of them are based on classification problem.

In [19], the authors have used the same five eclipse dataset for their ensemble machine learning models using 15 process metrics and number of faults. They

have presented their results in form of AAE as performance measure which we used with our results for comparison. The following table shows our hybrid selection and process metrics selection results along with results of [19].

TABLE 4.8: Results Validation

Datasets	**Our Results (AAE)**		**Results in [19] (AAE)**	
	Selected Hybrid(18)	**Selected Process(7)**	**LRCR Ensemble**	**GRCR Ensemble**
Eclipse-jdt	**0.38**	0.40	0.55	0.42
Equinox	**0.54**	0.55	0.61	0.6
Lucene	0.18	0.16	0.15	0.09
Mylyn	0.25	0.26	0.19	0.2
Eclipse-pde	0.32	0.32	0.2	0.22

The results show that our selection of features and machine learning (ML) model clearly outperforms in Eclipse-jdt and Equinox datasets not only in terms of process metrics, but the hybrid selection further improves the results and reduce error in prediction.

For Lucene, Mylyn and Eclipse-pde, our results do not show improvement in comparison to results presented by [19]. The reason behind this is that in the target paper, the authors have used minority oversampling techniques like SMOTER to balance their data and increase the number of non-zero instances, whereas in our experiments we have not engineered the datasets to preserve data credibility and produce accurate results.

Our results not only prove the importance and effectiveness of feature selection process for software metrics but also show the significance of selected process and code metrics which performed better as compared to all process and code metrics used in this study.

4.7 Discussions and Implications

In the field of software fault prediction, software metrics play a vital role in identifying loopholes and discrepancies. Extracting all relevant metrics and applying them to extract useful information is a tiresome process. A handsome amount of effort related to SFP is spent in obtaining and utilizing software metrics. Such issues require counter measures like feature selection to limit the amount of metrics which are particularly important for SFP process. Researchers have worked in the field of feature selection for software fault prediction but most of the research is focused on classification of faults using code metrics.

The findings of our research suggest that process metrics play an important role in fault prediction. They are equally informative and perform very well specially when combined with code metrics for fault prediction. By using feature selection, fault prediction process can be turned efficient and inexpensive in terms of resources. The hybrid feature set which was collected in this research by using wrapper subset selection, provides better results for prediction of number of faults when compared with all feature set. Utilizing a small amount of more significant features can make the SFP process not only cost effective but also more efficient accurate. We have also investigated the performance of different types of metrics against their selected subsets and the impact of different categories of software metrics in comparison with each other. Moreover, performance of each of the selected metrics is also presented.

Our research covers the aspects of feature selection for number of faults prediction. We have compiled and compared results of selected features, metrics categories and individual metric importance but there are also boundaries to our research which can be explored further to improve and extend our research. For example, our experiments were performed using five datasets. The number of instances in these datasets is quite small. If we can find and add more relevant datasets to our experimentation, results may improve to some extent. Investigating the results using more accurate machine learning ensembles can also enhance the output of this research.

Moreover, following are the answers of our research questions that were mentioned in Chapter 1. These answers have been identified after doing literature review and experimentation.

RQ1: What is the performance of process metrics as compared to code metrics in software fault prediction?

Our results indicate that the process metrics provided better results than the code metrics for number of software faults prediction. Process metrics yielded better MSE results than code metrics. The comparison of these results can be seen in section 4.3.

RQ2: Which process and code metrics subsets are most important in terms of performance for prediction of number of faults?

According to our findings, the subsets of process and code metrics given in section 4.1 are the most influential and significant subsets of metrics for predicting count of faults in software fault predictions which were selected using wrapper feature selection. The selected metrics from both types were added to form hybrid metrics subset and this hybrid subset of process and code metrics performed remarkably better than all metrics set. The hybrid subset showed better results as compared to all metrics set in all five datasets. The results given in section 4.2 can justify this finding.

RQ3: What is the effectiveness of different categories and individual selected process and code metrics?

The comparison of categories of selected metrics showed that change based category of process metrics outperformed other process metric categories and in code metrics categories, all categories performed equally well with method call based, cohesion based and access based categories having slightly better performance than rest of code metric categories. Individually, number of Authors and number of Version performed better in selected process metrics and fanout performed best in selected code metrics.

Chapter 5

Conclusion and Future Work

In the previous chapter, we have elucidated the details about the results that have been obtained by applying the proposed methodology and also these results are discussed in detail. In this chapter, we summarize our work and present a conclusion of the research we performed. As well as, we have identified some of probable directions for future research in this area.

5.1 Conclusion

The findings of our research include the investigation and selection of most important process and code metrics. We have done this selection using wrapper-based feature subset selection method. The selection was performed for finding number of software faults. We used five eclipse datasets for training and testing our machine learning ensemble model. For modeling we used random forests with 10-fold cross validation to produce accurate and generalized results as many researchers have claimed random forest to produce better results as compared to other traditional machine learners.

For improving number of faults prediction process and our results, we added selected process and code metrics to form a set of hybrid metrics composed of significant process and code metrics. Furthermore, for comparison we combined all

process and all code metrics used in our research, and compared the results of experiments on all metrics to selected hybrid metrics results. The hybrid metrics provided better results as compared to all metrics. Another finding of our research is that the selected process metrics set showed better performance than selected code metrics set.

We also investigated the significance of metric categories which showed that change based metrics from process metric categories presented better results as compared to other two categories of process metrics. For code metric categories, coupling based category gave better results than structure based category. Later on, when evaluating single metric significance, our results showed that number of versions and number of authors from selected process metrics provided better results and fanout presented better results in selected code metrics.

To conclude our research, we can say that the set of hybrid features which were identified in our study proved to be very significant and effective to predict the number of software faults with more precision as compared to regular set of metrics. Using this hybrid set, we can reduce the cost of collection of metrics as well as add efficiency and accuracy to prediction process.

5.2 Future Work

To further enhancement and vast the area of our research, we can add more bigger (larger number of instances) and diverse (related to different software) datasets to testing and experimentation process in the future. Besides, to improve further, we can evaluate the performance of the new ensemble methods as compared to our results. Moreover, in the future, we can also study the impact of class balancing on prediction of number of software faults as it is usually observed that fault prediction datasets are hugely imbalanced and number of instances in zero or no defect class tend to dominate. In addition to that, we can also evaluate the effect of different types of severity of bugs and impact of specific bug severity classes on fault prediction.

Bibliography

[1] Z. Xu, J. Liu, Z. Yang, G. An, and X. Jia, "The impact of feature selection on defect prediction performance: An empirical comparison," in *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2016, pp. 309–320.

[2] T. J. McCabe, "A complexity measure," *IEEE Transactions on software Engineering*, no. 4, pp. 308–320, 1976.

[3] F. B. Abreu and R. Carapuça, "Object-oriented software engineering: Measuring and controlling the development process," in *Proceedings of the 4th international conference on software quality*, vol. 186, 1994.

[4] S. R. Chidamber and C. F. Kemerer, "Towards a metrics suite for object oriented design," in *Conference proceedings on Object-oriented programming systems, languages, and applications*, 1991, pp. 197–211.

[5] M. H. Halstead *et al.*, *Elements of software science*. Elsevier New York, 1977, vol. 7.

[6] L. Briand, P. Devanbu, and W. Melo, "An investigation into coupling measures for c++," in *Proceedings of the 19th international conference on Software engineering*, 1997, pp. 412–421.

[7] J. M. Bieman and B.-K. Kang, "Cohesion and reuse in an object-oriented system," *ACM SIGSOFT Software Engineering Notes*, vol. 20, no. SI, pp. 259–262, 1995.

[8] J. Bansiya and C. G. Davis, "A hierarchical model for object-oriented design quality assessment," *IEEE Transactions on software engineering*, vol. 28, no. 1, pp. 4–17, 2002.

[9] J. C. Munson and S. G. Elbaum, "Code churn: A measure for estimating the impact of code change," in *Proceedings. International Conference on Software Maintenance (Cat. No. 98CB36272)*. IEEE, 1998, pp. 24–31.

[10] T. L. Graves, A. F. Karr, J. S. Marron, and H. Siy, "Predicting fault incidence using software change history," *IEEE Transactions on software engineering*, vol. 26, no. 7, pp. 653–661, 2000.

[11] Z. Xu, J. Liu, X. Luo, Z. Yang, Y. Zhang, P. Yuan, Y. Tang, and T. Zhang, "Software defect prediction based on kernel pca and weighted extreme learning machine," *Information and Software Technology*, vol. 106, pp. 182–200, 2019.

[12] T. Gyimothy, R. Ferenc, and I. Siket, "Empirical validation of object-oriented metrics on open source software for fault prediction," *IEEE Transactions on Software engineering*, vol. 31, no. 10, pp. 897–910, 2005.

[13] T. Zimmermann, R. Premraj, and A. Zeller, "Predicting defects for eclipse," in *Third International Workshop on Predictor Models in Software Engineering (PROMISE'07: ICSE Workshops 2007)*. IEEE, 2007, pp. 9–9.

[14] K. Gao and T. M. Khoshgoftaar, "A comprehensive empirical study of count models for software fault prediction," *IEEE Transactions on Reliability*, vol. 56, no. 2, pp. 223–236, 2007.

[15] H. Zhang, "An investigation of the relationships between lines of code and defects," in *2009 IEEE International Conference on Software Maintenance*. IEEE, 2009, pp. 274–283.

[16] M. Jureczko and D. Spinellis, "Using object-oriented design metrics to predict software defects," *Models and Methods of System Dependability. Oficyna Wydawnicza Politechniki Wrocławskiej*, pp. 69–81, 2010.

[17] E. Erturk and E. A. Sezer, "Iterative software fault prediction with a hybrid approach," *Applied Soft Computing*, vol. 49, pp. 1020–1033, 2016.

[18] R. A. Fagundes, R. M. Souza, and F. J. Cysneiros, "Zero-inflated prediction model in software-fault data," *IET Software*, vol. 10, no. 1, pp. 1–9, 2016.

[19] S. S. Rathore and S. Kumar, "Towards an ensemble based system for predicting the number of software faults," *Expert Systems with Applications*, vol. 82, pp. 357–382, 2017.

[20] R. Shatnawi, "The application of roc analysis in threshold identification, data imbalance and metrics selection for software fault prediction," *Innovations in Systems and Software Engineering*, vol. 13, no. 2-3, pp. 201–217, 2017.

[21] S. S. Rathore and S. Kumar, "Linear and non-linear heterogeneous ensemble methods to predict the number of faults in software systems," *Knowledge-Based Systems*, vol. 119, pp. 232–256, 2017.

[22] S. R. Aziz, T. Khan, and A. Nadeem, "Experimental validation of inheritance metrics' impact on software fault prediction," *IEEE Access*, vol. 7, pp. 85 262–85 275, 2019.

[23] M. Rizwan, A. Nadeem, and M. A. Sindhu, "Empirical evaluation of coupling metrics in software fault prediction," in *2020 17th International Bhurban Conference on Applied Sciences and Technology (IBCAST)*. IEEE, 2020, pp. 434–440.

[24] Ö. F. Arar and K. Ayan, "Software defect prediction using cost-sensitive neural network," *Applied Soft Computing*, vol. 33, pp. 263–277, 2015.

[25] S. Matsumoto, Y. Kamei, A. Monden, K.-i. Matsumoto, and M. Nakamura, "An analysis of developer metrics for fault prediction," in *Proceedings of the 6th International Conference on Predictive Models in Software Engineering*, 2010, pp. 1–9.

[26] L. Madeyski and M. Jureczko, "Which process metrics can significantly improve defect prediction models? an empirical study," *Software Quality Journal*, vol. 23, no. 3, pp. 393–422, 2015.

[27] S. S. Rathore and S. Kumar, "A decision tree regression based approach for the number of software faults prediction," *ACM SIGSOFT Software Engineering Notes*, vol. 41, no. 1, pp. 1–6, 2016.

[28] C. Manjula and L. Florence, "Deep neural network based hybrid approach for software defect prediction using software metrics," *Cluster Computing*, vol. 22, no. 4, pp. 9847–9863, 2019.

[29] I. H. Laradji, M. Alshayeb, and L. Ghouti, "Software defect prediction using ensemble learning on selected features," *Information and Software Technology*, vol. 58, pp. 388–402, 2015.

[30] H. Wang, T. M. Khoshgoftaar, and A. Napolitano, "A comparative study of ensemble feature selection techniques for software defect prediction," in *2010 Ninth International Conference on Machine Learning and Applications*. IEEE, 2010, pp. 135–140.

[31] T. M. Khoshgoftaar, K. Gao, and N. Seliya, "Attribute selection and imbalanced data: Problems in software defect prediction," in *2010 22nd IEEE International conference on tools with artificial intelligence*, vol. 1. IEEE, 2010, pp. 137–144.

[32] T. M. Khoshgoftaar and K. Gao, "Feature selection with imbalanced data for software defect prediction," in *2009 International Conference on Machine Learning and Applications*. IEEE, 2009, pp. 235–240.

[33] R. S. Wahono and N. Suryana, "Combining particle swarm optimization based feature selection and bagging technique for software defect prediction," *International Journal of Software Engineering and Its Applications*, vol. 7, no. 5, pp. 153–166, 2013.

[34] H. Wang, T. M. Khoshgoftaar, J. Van Hulse, and K. Gao, "Metric selection for software defect prediction," *International Journal of Software Engineering and Knowledge Engineering*, vol. 21, no. 02, pp. 237–257, 2011.

[35] C. Catal and B. Diri, "Investigating the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problem," *Information Sciences*, vol. 179, no. 8, pp. 1040–1058, 2009.

[36] C. Ni, W.-S. Liu, X. Chen, Q. Gu, D.-X. Chen, and Q.-G. Huang, "A cluster based feature selection method for cross-project software defect prediction," *Journal of Computer Science and Technology*, vol. 32, no. 6, pp. 1090–1107, 2017.

[37] A. Okutan and O. T. Yıldız, "Software defect prediction using bayesian networks," *Empirical Software Engineering*, vol. 19, no. 1, pp. 154–181, 2014.

[38] P. He, B. Li, X. Liu, J. Chen, and Y. Ma, "An empirical study on software defect prediction with a simplified metric set," *Information and Software Technology*, vol. 59, pp. 170–190, 2015.

[39] A. Kalsoom, M. Maqsood, M. A. Ghazanfar, F. Aadil, and S. Rho, "A dimensionality reduction-based efficient software fault prediction using fisher linear discriminant analysis (flda)," *The Journal of Supercomputing*, vol. 74, no. 9, pp. 4568–4602, 2018.

[40] H. Turabieh, M. Mafarja, and X. Li, "Iterated feature selection algorithms with layered recurrent neural network for software fault prediction," *Expert systems with applications*, vol. 122, pp. 27–42, 2019.

[41] M. D'Ambros, M. Lanza, and R. Robbes, "An extensive comparison of bug prediction approaches," in *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*. IEEE, 2010, pp. 31–41.

[42] T. Wang, W. Li, H. Shi, and Z. Liu, "Software defect prediction based on classifiers ensemble," *Journal of Information & Computational Science*, vol. 8, no. 16, pp. 4241–4254, 2011.

[43] C. Sammut and G. I. Webb, *Encyclopedia of machine learning*. Springer Science & Business Media, 2011.

[44] C. J. Willmott and K. Matsuura, "Advantages of the mean absolute error (mae) over the root mean square error (rmse) in assessing average model performance," *Climate research*, vol. 30, no. 1, pp. 79–82, 2005.

[45] R. Kohavi, G. H. John *et al.*, "Wrappers for feature subset selection," *Artificial intelligence*, vol. 97, no. 1-2, pp. 273–324, 1997.