# An Investigation of the Accuracy of Code and Process Metrics for Defect Prediction of Mobile Applications

ARVINDER KAUR, KAMALDEEP KAUR , HARGUNEET KAUR

*USICT*

*GGS,Indraprastha University*

*Sec-16C ,Dwarka*

*Delhi,INDIA*

[1]arvinderkaurtakkar@ yahoo.com

[2]kdkaur99@gmail.com

[3]harguneet.sethi@gmail.com

***Abstract*— Mobile applications have been around for a long time and proved to be a new excited market where everyone want to engage themselves. They have become more important than webpages nowadays. Companies are giving more preference to mobile apps as compared to websites because of their user friendliness , better visibility and ease of social networking. This paper compares static code metrics and process metrics for predicting defects in an open source mobile applications. Correlation coefficient, mean absolute error and root mean squared error with process metrics as predictors are significantly better than with code metrics as predictors. Also the combined model based on process and code metrics is better than the model based on code metrics. It is shown that process metrics based defect prediction models are better for mobile applications in all 7 machine learning techniques used for modelling.**

***Keywords*— Code metrics, process metrics, software metrics, defect prediction,mobile apps**

## I. INTRODUCTION

Testing step in software process consumes huge amount of cost and effort. In testing phase , all the modules are uniformly tested but it is to be noticed that bugs are not uniformly distributed hence we should apply some method to save the resources. If modules with high number of bugs are identified before formal testing phase, then it will be easy for the testers to test only those modules with high priority.Nowadays people are increasingly involved in app downloads and users just abandon any mobile app if they find any functionality problem. To stay competitive companies need to build robust mobile apps. Thus for mobile apps it is a challenge to determine which components need to be tested[14-17] so that no critical steps are missed. Users just expects that all apps should function correctly, quickly and consistently and it doesn't crash , hang or stall.

There is a transition from desktop to web and now web to mobile. Desktop apps are targeted to specific platform which is easy to access where as web based apps become a bit challenging by as they may be accessed across different browsers and mobile apps have to handle the complexity of various hardware devices and software platforms, for example , Android, BlackBerry, windows, iOS etc.This makes mobile app testing to a challenging task to perform. Mobile apps should adapt themselves to different screen sizes. Interaction of user to desktop and web applications are limited to keyboard and mouse where as mobile apps make user interaction as fluid as possible. Mobile tester have to ensure that the application performs efficiently when user interacts in different ways. Mobile apps depend on emulators and simulators which have their own capabilities and limitations. Mobile apps are installed, removed or updated more frequently than desktop applications.Platform makes a difference to software development and maintenance process.There are many previous studies that propose defect prediction models based on static code metrics[[1],[2],[4],[8],[9],[13]]. Only few studies are based on process and code metrics both[[6],[11],[12]]. However, most of the studies that evaluate both code and process metrics are based on proprietary and open source general applications. To the best of our knowledge, there are very few studies based on mobile applications[11]. To the best of our knowledge, this current study may be the first attempt to investigate process metrics for defect prediction of mobile application. The main aim of this research is to investigate both code and process metrics for defect prediction of mobile applications .In this study, code metrics are extracted by ckjm and process metrics are extracted by analysing the change history of source code. These metrics are taken as input and modelling techniques are applied to predict fault prone components.

The rest of the paper is organised as follows: In section 2 related work is presented. Section 3 provides the description of studied metrics which explains code metrics in section 3.1 and process metrics in section 3.2. Section 4 gives the description of how data is collected and experiment is setup. Results are presented in section 5. Section 6 discusses the threats to validity. Conclusion and future work are presented in Section 7.

## II. Related Work

Number of studies [[6],[11],[12]] have been done by researchers on code and process metrics for defect prediction models but a very few studies are done to identify the fault prone areas of mobile applications.

SPINELLIS et al.[8] extracted the object oriented metrics using metric calculation tool ckjm. The main goal of their study is to construct software defect prediction models. Software metrics of several open source and proprietary Java applications were taken as model input and number of defects as output. It was observed that the classes which were selected actually have defects on higher level. Models of their paper selected those Java classes which had 80% or more of the total defects.

Marian Jureczko [6] performed empirical analysis on importance of different code and process metrics in defect prediction models of desktop applications. He conducted his experiment on 48 releases of 15 open source projects and 38 releases of 7 proprietary projects. For each of the code and process metric, correlation coefficient with number of defects were calculated and analysis was performed after defect prediction models were built using linear stepwise regression. It was concluded that process metrics showed high value on various model evaluation measures such as correlation coefficient. To the best of our knowledge process metrics of mobile applications have not been extracted.

Raimund Moser et al.[12] conducted implementation on two set of metrics: change and static code metrics for eclipse project where it was concluded whether file is defective or not. Three models are built by Moser el al.[12] for their study : Naive Bayes, decision tree and logistic regression. Cost sensitive classification was performed for predicting errors which was proved very successful. Process metrics for the eclipse data showed results more efficiently for predicting defects than code metrics.

Lech Madeyski et al.[7] investigated process metrics in defect prediction. Process metrics taken in their study were: Number of Different Committers, Number of Modified Lines, Is New Defect, Number of Defects in Previous Revision and Number of Revisions. Usage and taxonomy of process metrics in defect predicting models is analysed. They performed their analysis on different datasets based on different research papers.

Ruchika et al.[11] investigated 15 evolutionary and hybridized evolutionary techniques for their predictive capability in software defect prediction on 5 open source systems based on static code metrics. It was concluded that evolutionary computing techniques are more powerful for prediction defects. The best technique was found to be GAS-ADI. The following points were also concluded:

a) Metrics which can be used to predict software defects in early stage of software lifecycle are: RFC, LCOM and LOC.

b) Evolutionary computation techniques shows better results with software engineering to predict defects. However Ruchika et al. [11] did not investigate process metrics of mobile applications and also they didn't compare the performance of static code metrics with process metrics.

Lech Madeyski et al.[10] performed empirical evaluation and identified the process metrics which can improve the defect prediction models based on code metrics. Prediction of the models which used code metrics are compared with the models which used product as well as process metrics. The results were significant with the dataset containing product metrics and additionally Number of Distinct Committers(NDC) as compare to the dataset which does not contain NDC when the effect size was minimum. Similarly Number of Modified Lines (NML) showed significant better results than the normal models without NML .It was concluded that NDC process metric can be recommended for building defect prediction models.

In this current paper, we study three defect prediction models: Code metric model, Process metric model and combined model of code and process metrics both for predicting defects in Android alarmclock mobile application.

## III. Studied Metrics

There are two types of metrics: Code and Process metrics. Code and process metrics are quantitative measures which are viewed separately and enable software practitioners to observe the efficiency of software products and process. Quality and productive data is collected which is analysed and compared with its past values and determine whether improvements have occured or not. Metrics have been calculated to look at the problematic areas so as to further improve it.

### *A. Code Metrics*

Code metrics are the metrics resulting from analysis of software code. Code metrics for Java code can be easily extractedby ckjm tool[3]. Metrics which are calculated by ckjm for each Java class are tabulated in Table 1.

TABLE I
CKJM METRIC DESCRIPTION

| **Metric** | **Description** |
|---|---|
| WMC | Weighted methods per class. It is the sum of the complexities of all the methods in a class |
| DIT | Depth of Inheritance Tree. This metric measures the inheritance levels from the object hierarchy top. . |
| NOC | Number of Children .It measures the number of immediate descendants of the class. |
| CBO | Coupling between object classes. This metric represents the count of number of other to which a given class is coupled |
| RFC | Response for a Class. The metric is the number of district methods and constructors that are called by a class. |
| LCOM | Lack of cohesion in methods. This metric is the object-oriented metric which counts the sets of methods in a class that are not related through the passing of some of the class's fields. |
| Ca | Afferent couplings. It determines the number of classes and interfaces from other package that use the specific class. |
| Ce | Efferent couplings. This metric is also known as Outgoing Dependencies measures the number of types inside a class that depend on types of other classes. |
| NPM | Number of Public Methods. The NPM metric counts all the methods in a class which are considered as public. |
| LCOM3 | Lack of cohesion in methods.LCOM3 varies between 0 and 2. |

| | |
|---|---|
| LOC | Lines of Code. This software metric measures the size of computer program by counting the lines from java binary code. |
| DAM | Data Access Metric. Data Access metrics measures the ratio of the private fields declared in the class to the total number of fields in the class. |
| MOA | Measure of Aggregation.It counts the number of data declaration that can be class fields also which are declared as user defined classes. |
| MFA | It is the ratio of number of methods inherited by a class to the number of methods accessible by member methods of the class |
| CAM | Cohesion Among Methods of Class. This metric evaluates the consistency of methods in a class using the parameter list of the methods. |
| IC | Inheritance Coupling. This metric refers to the coupling of two class when one class is the subclass of other. |
| CBM | Coupling Between Methods. The metric measure the total number of new/redefined methods to which all the inherited methods are coupled. |
| AMC | Average Method Complexity. This metric refers to the average method size for each class. |
| CC | McCabe's cyclomatic complexity. It refers to the number of independent paths in a method (function) plus one. |
| AC Count | It is the average of The McCabe's cyclomatic complexity which is calculated for each method. |
| MAX count | It is the maximum value of CC among all the methods. |

### *B. Process Metrics*

Process metrics[5] enable software practitioners to analyse the existing process by collecting data across all the revisions and over long period of time.Process metrics considered in our study are those which are revelant to empirical studies in software engineering. The main motivation is to improve the software process. It measures the specific attributes of the process and create a set of meaningful metrics. These metrics help to make strategic decisions for improvement of the process and about how to complete software process framework. Process metric like 'Performance of each phase of process' is not included in our study because open source software repositories doesnot follow waterfall lifecycle model. Process metrics are set down manually.

Following metrics are recorded from the comments by different authors for the particular revision of the dataset: LOC_ADDED: Number of lines added. This metric refers to the number of lines added in the specific revision for each class. It is the sum of lines added for all the changes occurred in a class for a specific revision of dataset.

LOC_DELETED: Number of lines deleted. This metric refers to the number of lines deleted in the specific revision for each class It is the sum of lines deleted for all the changes occurred in a class for a specific revision of dataset.

NA: Number of Authors. This metric refers to the number of distinct authors that checked or changed a particular file or class into the repository for the particular revision of dataset. Author has may be corrected the bug or implemented the change or just committed the changes and implementation.

NR: Number of Revisions. This metric calculates the number of times the class is referred by the authors for the change or bug or some enhancements.

NB: Number of Bugs. This is the most important metric which is recorded as number of times a particular class in involved for bug-fixes. This metric mainly helps in defect prediction as it is taken as the dependent variable during modelling.

## IV. DATA COLLECTION AND EXPERIMENTAL SETUP

In our study, code and process metrics of a mobile application have been used to predict post- release defects for the Android release 1.6 alarmclock application package which has been taken from the Android platform. It is available for download in the github repository(https://github.com). Data is collected from the github repository for code metrics on 16 Feb,2015 and for process metrics on 20 April,2015.The research aim here is to determine whether the defect prediction techniques which are applicable to desktop and web applications can be adapted to mobile apps or not. The flow diagram is demonstrated in figure 1 shows how to build defect prediction models. Source code and development process data are extracted from the github repository and mapped defects to source code classes from the comments posted by the authors. After extracting source code, it is synthesized in Android application development environment. Thereafter, 20 source code metrics listed in Table 1 have been extracted using ckjm tool. Process metrics are also extracted from the comments posted by the authors in github repository for any change or defect in the Java class files of alarmclock package. Thus 5 process metrics which have been extracted from github repository. Only Java files are considered, all other files are skipped because ckjm calculates metrics of Java class files, therefore process metrics of only Java files are recorded for comparative study. The data format of the metrics so obtained are in numerical form which has been tabulated in our relational table. Once source code metrics and process metrics are calculated, modelling is performed using Weka tool . In this study we also analyse the relationship between process metrics and number of defects in each class files. Table 2 shows the summary of data and class files with total number of bugs recorded.

We have determined both code and process metrics which are useful for predicting defects in mobile application.Prediction is done by dividing the collected dataset in training and testing set.We used leave one out cross validation method. Various modelling techniques have been applied to these metrics for predicting the defects like Linear Regression Model, RBFNetwork, Simple Linear Regression, SMOreg, IBk, Kstar, LWL.Table 3 gives the brief overview of all the techniques used.We first consider code metrics , process metrics separately in our analysis and also the combination of both. We have compared all the three models with the one based on code metrics, process metric and also the combination of two.Other researchers may have used slightly different set of process metrics[12].In previous research on process metrics based defect prediction, only statistical modelling techniques have been used. Machine learners have not been investigated in-depth.

TABLE 2
SUMMARY OF THE ANDROID ALARMCLOCK PACKAGE

| Release | #Class Files | Metrics | Number of Bugs |
|---|---|---|---|
| 1.6 (Fri Sep 18 2009) | 16 | 20 source code and 5 proces | 41 |

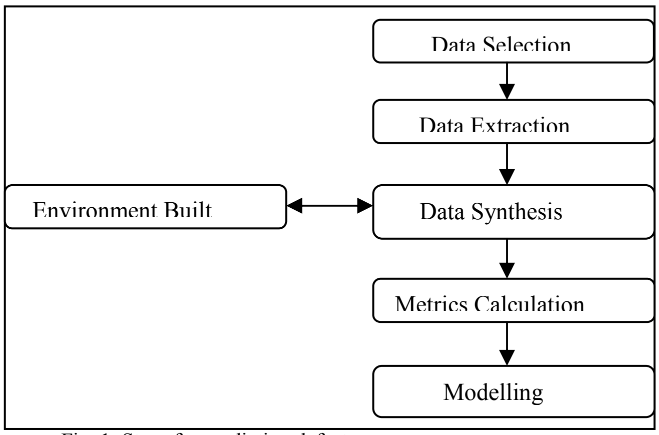

Fig. 1 Steps for predicting defects

TABLE 3
MODELLING TECHNIQUES USED

| Modelling Technique | Description |
|---|---|
| Linear Regression Model | It uses the Akaike criterion for model selection and deals with weighted instances. It deals with many variables. |
| RBF Network | K-means clustering algorithm is used to implement a normalized Gaussian radial basis function network. |
| Simple Linear Regression | It picks the value which results in lowest squared error. Missing values are not allowed. |
| SMOreg | This technique uses the support vector machine for regression. The parameters can be extracted using various algorithms. The algorithm is selected by setting the RegOptimizer. |
| IBk | It selects appropriate value of k based on cross-validation. |
| Kstar | This is the instance based classifier i.e. class of training instances determines the class of the test instance. It uses entropy based distance function. |
| LWL | It is Locally weighted learning. It uses instance-based algorithm to assign instance weights It has the capability to perform classification or regression. |

It is hypothesized that, the classes which are defect prone have high number of modifications(additions and deletions),high revision number, number of distinct authors is large ,number of bug fixes in a particular revision is high.

The independent variables used in this study are all code and process metrics which are extracted and dependent variable used in the whole experiment is number of bugs in each Java class file. Process metrics are recorded from the comment history by software developers. It is to be seriously noted that the aim of this research is not to determine the best modelling technique for defect classification on mobile application dataset but to find the best set of metrics which help in predicting defect proneness class files of mobile application. To decide best set of metrics, we consider following:

- Code metrics are analysed in Weka tool with respect to number of bugs.
- Process metrics are analysed in Weka tool with respect to number of bugs.
- Combination of both code and process metrics are analysed. Results are then compared that which set of metrics are better as defect predictors.

## V. RESULTS

The first step we performed to get our results is building three models, one using only source code metrics referred to as code model, other using process metrics called as process model and third one using both type of metrics referred to as combined model for predicting high defect prone files. Since the performance of the three models also depends on machine learner used for modelling, we explore 7 machine learning techniques. Seven machine learning techniques used are: Linear Regression Model, RBFNetwork, Simple Linear Regression, SMOreg, IBk, Kstar, LWL We select these machine learning techniques because these techniques have been found to be useful for defect prediction[18]. Comparison of the defect prediction models is done in terms of accuracy measures such as Correlation coefficient, Mean absolute error and Root mean squared error. Android alarmclock package release 1.6 dataset is taken, details of which were presented Table 2 section 4. Table 4 displays the value of correlation coefficient of process metrics, code metrics and combination of two. Accurate models should attain high value of correlation coefficient. It is visible from the graph of figure 2 that in all the seven machine learning techniques applied, process metrics have higher value than code metrics for Correlation coefficient. The value of this measure should be close to one. It may be noted that even combination of both process and code metrics model attains less value than process metrics but high value than code metrics on most of the techniques.
Mean absolute error is also calculated from Weka tool in which process metrics performs significantly better than code metrics and combination of two as shown graphically in figure 3. Modelling techniques should have low value of mean absolute error. Table 5 displays the values of mean absolute

error with different techniques for process metrics model, code metrics model as well as combined model .

Table 6 displays values of root mean squared error for process metrics model , code metrics model and combined model with various modelling techniques.For simple linear regression modelling technique process metrics and combined metrics shows the same result for root mean squared error but it seems not to be worth to collect code metrics in addition to process metrics. Root mean squared error is shown graphically in Fig 4 for process metrics, code metrics and combination of two using various modelling techniques

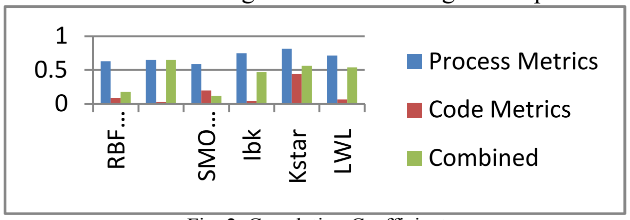

Fig. 2 Correlation Coefficient

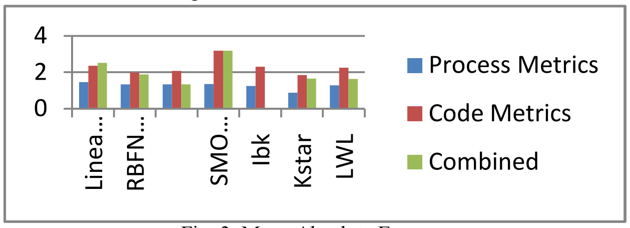

Fig. 3 Mean Absolute Error

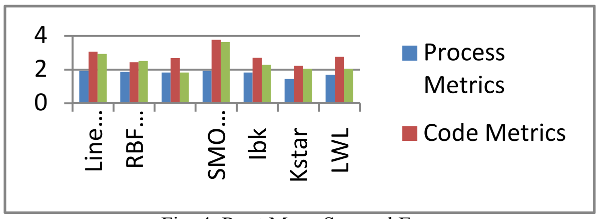

Fig. 4 Root Mean Squared Error

Overall it is observed that process metrics outperforms the code metrics for predicting defects. The combined model performs similar results to process metrics in some techniques but not significantly better. For detecting defects it seems that code metrics is a weaker set than process metrics. Classes with higher value of bug-fixes i.e. larger value of number of bugs are defective which may mean that bug elimination activities tend to introduce new defects. This point is valuable for understanding the construction of model for defect prediction for mobile applications.

Thus we have observed from the above figures and Tables that:

1) For correlation coefficient, Kstar techniques shows better results with value 0.8128 than all other techniques for process metrics.

2) Mean absolute error is measured for all the techniques but for Kstar modelling techniques it shows far better results than all other with value of 0.8857 for process metrics.

3) Similarly root squared error shows better results with Kstar modelling technique for process metrics of value 1.4497. From these results it is concluded that Kstar modelling technique outperforms all other techniques for defect prediction of mobile applications.

## VI. Threats to Validity

Android specific environment is conducted for getting the results of our study. However researchers have made the number of observations for defect prediction for various datasets but no one has conducted their study on mobile application dataset. We hope that our experiment strengthens the existing body of knowledge for future theory of defect prediction.

Our study uses alarmclock application package of android for version 1.6.We may have included other application packages for calculating metrics in defect prediction. The fact that we use simple techniques for modelling give us some confidence about our results which may obtain different results when sophisticated techniques are used.We have used limited code and process metrics for detecting defect prone areas. There are various hidden complex metrics which can be the powerful defect predictors. Only java class files are used to calculate metrics for the android package. Other files are excluded in our study. This may have skipped many files which may resulted in abnormal values of some metrics.Efficiency and effectiveness are process metrics which are difficult to obtained from open source software repositories.

There may be several reasons for erroneous data: data obtained from repository, mapping between source files and defects, extraction of code and process metrics. In addition process metrics are extracted from Android github repository which may contain some errors. Defect prone areas are observed from the Android package of single version1.6.Other versions of same package may be included for getting more confidence about the results.

## VII. Conclusion and Future Work

The results of this study identify defect predictors using process metrics and code metrics which can be extracted from github repository. A set of 20 code metrics and 5 process metrics are used to generate results of the Android release 1.6 project. The results have shown that process metrics outperforms the code metrics for defect prediction models.

The findings of our research included mobile application dataset which is used for the first time from which process and code metrics both are extracted. Prediction models are built to identify the class files which are defective or which are not. According to the developer a class file is free of errors but if it undergoes many changes then atleast one defect is introduced which is detected by the predictors while doing modelling. Future work will focus on following issues for defect prediction:

1) Java class files are included to calculate metrics. In future we may include files from other programming language to calculate code and process metrics.

2) Single version 1.6 of android application has been taken which can be extended to all the versions of a particular application. It may results in more promising values.

3) In this study dataset is taken for a single application of android i.e. alarmclock. Other applications of Android will also be considered in future.

## REFERENCES

[1] D'Ambros, Marco, Michele Lanza, and Romain Robbes. "An extensive comparison of bug prediction approaches." *Mining Software Repositories (MSR), 2010 7th IEEE Working Conference on*. IEEE, 2010.
[2] S. Chidamber, and C. Kemerer, "A metrics suite for object oriented design", IEEE Transactions on Software Engineering, vol. 20, no. 6, pp.476-493, 1994
[3] ckjm — Chidamber and Kemerer Java Metrics - http://www.spinellis.gr/sw/ckjm/
[4] K.E. Emam, W. Melo, and J.C. Machado, "The prediction of faulty classes using object-oriented design metrics", The Journal of Systems and Software, pp. 63-75, 2001
[5] Graves, T. L., Karr, A. F., Marron, J. S., Siy, H. 2000. Predicting fault incidence using software change history. IEEE Transactions on Software Engineering, 26(7): 653 – 661 (July 2000).
[6] Jureczko, Marian. "Significance of different software metrics in defect prediction." Software Engineering: An International Journal 1.1 (2011): 86-95.
[7] Jureczko, Marian, and Lech Madeyski. "A review of process metrics in defect prediction studies." Metody Informatyki Stosowanej 5 (2011): 133-145.
[8] Jureczko, Marian, and Diomidis Spinellis. "Using object-oriented design metrics to predict software defects." Models and Methods of System Dependability. Oficyna Wydawnicza Politechniki Wrocławskiej (2010): 69-81.
[9] O'Keeffe, Mark, and Mel O. Cinnéide. "Search-based software maintenance." *Software Maintenance and Reengineering, 2006. CSMR 2006. Proceedings of the 10th European Conference on*. IEEE, 2006: pp. 10-pp.
[10] Madeyski, Lech, and Marian Jureczko. "Which process metrics can significantly improve defect prediction models? An empirical study." *Software Quality Journal* (2014): 1-30.
[11] Malhotra, Ruchika, Nakul Pritam, and Yogesh Singh. "On the applicability of evolutionary computation for software defect prediction." *Advances in Computing, Communications and Informatics (ICACCI, 2014 International Conference on*. IEEE, 2014,2249-2257.
[12] Moser, Raimund, Witold Pedrycz, and Giancarlo Succi. "A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction." *Software Engineering, 2008*,181-190.
[13] Y. Singh, A. Kaur, and R. Malhotra, "Empirical validation of object oriented metrics for predicting fault proneness models", Software Quality Journal, pp. 3-35,2010.
[14] Muccini, Henry, Antonio Di Francesco, and Patrizio Esposito. "Software testing of mobile applications: Challenges and future research directions." *Automation of Software Test (AST), 2012 7th International Workshop on*. IEEE,2012,pp:29-35.
[15] Franke, Dominik, and Carsten Weise. "Providing a software quality framework for testing of mobile applications." *Software Testing, Verification and Validation (ICST), 2011 IEEE Fourth International Conference on*. IEEE, 2011.pp: 431-434.
[16] Kirubakaran, B., and V. Karthikeyani. "Mobile application testing—Challenges and solution approach through automation." *Pattern Recognition, Informatics and Mobile Engineering (PRIME), 2013 International Conference on*. IEEE, 2013.pp:79-84
[17] Kirubakaran, B., and V. Karthikeyani. "Mobile application testing—Challenges and solution approach through automation." *Pattern Recognition, Informatics and Mobile Engineering (PRIME), 2013 International Conference on*. IEEE, 2013.pp:555-560.
[18] Arvinder Kaur, Kamaldeep Kaur: An Empirical Study of Robustness and Stability of Machine Learning Classifiers in Software Defect Prediction. ISI (1) 2014: 383-397

## APPENDIX

Tables 4, 5 and 6 are shown as follows:

TABLE 4
CORRELATION COEFFICIENT

| Techniques | RBF | Simple Linear reg | SMO reg | Ibk | Kstar | LWL |
|---|---|---|---|---|---|---|
| Process Metrics(LOC_ADDED,LOC_DELETED,NA,NR,NB) | 0.631 | 0.649 | 0.589 | 0.748 | 0.813 | 0.714 |
| Code Metrics(Mentioned in section III) | 0.087 | 0.026 | 0.199 | 0.040 | 0.439 | 0.067 |
| Combined(both code and process metrics) | 0.181 | 0.649 | 0.120 | 0.470 | 0.564 | 0.543 |

TABLE 5
MEAN ABSOLUTE ERROR

| Techniques | Linear Reg Model | RBF | Simple Linear Reg | SMO reg | Ibk | Kstar | LWL |
|---|---|---|---|---|---|---|---|
| Process Metrics(LOC_ADDED,LOC_DELETED, NA,NR,NB) | 1.469 | 1.335 | 1.342 | 1.349 | 1.25 | 0.886 | 1.289 |
| Code Metrics(Mentioned in section III) | 2.368 | 1.999 | 2.084 | 3.185 | 2.312 | 1.845 | 2.259 |
| Combined(both code and process metrics) | 2.517 | 1.879 | 1.342 | 3.191 | 1.813 | 1.649 | 1.634 |

TABLE 6
ROOT MEAN SQUARED ERROR

| Techniques | Linear Reg | RBF Ntwk | Simple Linear | SMO reg | Ibk | Kstar | LWL |
|---|---|---|---|---|---|---|---|
| Process Metrics(LOC_ADDED,LOC_DELETED, NA,NR,NB) | 1.933 | 1.869 | 1.829 | 1.9252 | 1.837 | 1.449 | 1.689 |
| Code Metrics(Mentioned in section III) | 3.063 | 2.441 | 2.6948 | 3.774 | 2.704 | 2.220 | 2.759 |
| Combined(both code and process metrics) | 2.934 | 2.519 | 1.829 | 3.6299 | 2.278 | 2.052 | 2.043 |