Predicting Change Proneness using Object-Oriented Metrics and Machine Learning Algorithms

Abdullah Al-Senayen, Abdurhman Al-Sahood
Computer Science, College of Computer Science & IT
King Faisal University
Al-Ahsa, Saudi Arabia
{aoaams, bo.dahm.99)@gmail.com

Mohammed Misbhauddin
Information Systems, College of Computer Science & IT
King Faisal University
Al-Ahsa, Saudi Arabia
mmisbhauddin@kfu.edu.sa

***Abstract*— Open Source Software (OSS) has become a huge part of today's software market and a good source for investments. The establishment of the "National Program for Free & Open Source Software Technology" by the top research center (KACST) in Saudi Arabia to encourage the use of OSS within the community is a major motivation to our work. OSS comes with numerous challenges, one of which is constant change. Being able to identify and measure the change proneness in open source software will ensure saving resources like time and effort. In this paper, we measure the capability of classes of machine learning algorithms to predict change proneness in OSS by using object-oriented metrics. Four classes of machine learning algorithms were considered: Probability-based, Function-based, Instance-based and Tree-based. One complete version of the OSS was used as a training set and tested on the subsequent version to predict the change. The machine learning algorithms were compared based on accuracy, specificity, sensitivity and root mean squared error. We found that nearest neighbor algorithm performed better than the other algorithms in terms of sensitivity and specificity. In the future, we plan to test with different parameters to find a better prediction model for software change proneness.**

Keywords-open source software; object-oriented; change proneness; maintainability; prediction.

I. INTRODUCTION

The concept of change is well-known in Software Engineering and the series of changes made to a software system is termed as Software Evolution [1]. The need for evolving software comes because of incorporating new functionality, modifying existing functionality or adapting to new environment conditions etc. However, the impact of this change on the whole system is based on the manner in which the project was developed. According to Güneş Koru and Liu [2], software development can either be closed source or open source. On one hand, closed source projects are well planned and executed. Hence, changes to such systems are localized and can be dealt with in a less haphazard manner. On the other hand, open source projects are developed in an evolutionary manner [3] as a result of which the changes are not restricted and scattered consistently throughout the classes in the project.

OSS has come a long way since the start of its movement in the 1970s. The vision of OSS has changed technology and its market forever. It was the cause of a huge number of breakthroughs. It gave us Google Android, Mozilla Firefox, Linux, Apache, and many more. As OSS changed the world, Saudi Arabia was not an exception. Although the OSS ecosystem in Saudi Arabia is young and developing, it is growing at a fast pace. One aspect of its growth is the huge efforts done by King Abdul-Aziz City for Science and Technology (KACST) actively working to promote the use of OSS in Saudi Arabia. It is running a number of international workshops on the uses of OSS and they are helping in developing standards, awarding innovations, and support academic research on the subject [4].

As OSS development grows in the kingdom and all around the world, we need to consider the characteristics of a good OSS [5]. Extensive research has been conducted over the years to study the relationship between software metrics and various software quality attributes like fault proneness and maintainability [2], [6]-[20]. Around 40-70% of entire cost of a software project is spent on maintenance [17]. The probability that part of software might change is usually referred to as change-proneness. Determining change-prone classes helps in software maintenance, ensuring corrective actions are initiated beforehand. Identifying these classes and the factors that cause these changes is major issue faced during software development. The factors that cause these changes as characterized by Arisholm and Briand [21] as:

1. Structural characteristics of classes (e.g., their coupling).
2. Coding quality of classes.
3. Factors that are captured by the defect history of the classes in the previous release.
4. Change Management team skill and expertise.

In order to identify the causes of change-proneness, we need to identify a rich set of metrics that cover the above-mentioned factors, and hence, help in identifying the exact factors that influence change. In this paper, we are going to look into measures that link the structural characteristics of the classes with their change proneness capability during development. In order to obtain empirical evidence, we analyzed a set of structural metrics and change data that belonged to an open-source project, Heretrix [22]. The change data was extracted comparing classes between consecutive releases of the object-oriented project and the object-oriented metrics from these releases. Metrics were

 ISBN: 978-1-61208-367-4

collected using tools such as ckjm (Chidamber and Kemerer Java Metrics Suite) [23], Dependency Finder [24] and the metric 1.3.6 plug-in in Eclipse [25]. Structural properties of classes, as measured by these metrics, are then associated with change-proneness. In this paper, we are going to treat the aspect of predicting change proneness. Our aim is to measure the capability of certain machine learning algorithms, to predict change proneness in open source software using object-oriented metrics. By being able to predict these classes, we will ensure saving resources like time, money and effort.

The rest of this paper is organized as follows. Section 2 comprises of a detailed literature survey of the various studies done in the past that relate structural properties of classes to their problem source such as change and defect proneness. Section 3 provides the pre-requisite used for the experiment including information on the open source project and the various object-oriented metrics and machine learning algorithms used. The Experimental Setup, Hypotheses and Results are provided in Section 4 and 5, respectively. Results from the experiment are then analyzed with respect to the stated hypotheses in Section 6. Threats that may have affected the validity of the results are highlighted in Section 7. Section 8 concludes the paper and emphasizes on scope for future work based on our findings.

II. RELATED WORK

Gyimothy et al. [6] used open-source software and object-oriented metrics to predict software faults comparing linear and logistic regression models against machine learning algorithms such as decision trees and neural networks. Van koten and Gray [7] used Bayesian Networks as the model to predict maintainability, which is quantified in their approach as the number of lines of changed during a 3-year period. They concluded that by using Bayesian Networks, the model could predict maintainability more accurately than regression-based models. Koru and Liu [2] constructed a tree-based model to predict change-proneness in two large open source projects. They suggested that practitioners should start collecting static metrics and change data to aid their maintenance effort. Zhou and Leung [8] used Multivariate Adaptive Regression Spline (MARS) to predict maintainability.

Eski and Buzluca [9] also used OO metrics to predict change-proneness and its effect on testing effort. Unlike our approach, they used data values from a single version of the software and concluded that change-proneness can be estimated correctly by selecting some optimal set of metrics. Khomh et al. [10] and Romano et al. [11] also proposed a change and fault prediction model but with assessing the impact of anti-patterns rather than OO metrics. Anti-patterns are code patterns with poor design choices.

Lu et al. [12] used statistical meta-analysis techniques to investigate the relationships between OO metrics and change-proneness. Elish and Khiaty [13] also used metrics to predict change-prone classes. In their work, multiple multivariate logistic regression models were built using different sets of dependent and independent variables. They concluded that prediction of change proneness is accurate when product metrics are combined with evolutionary metrics.

Peer and Malhotra [14] used Adaptive Neuro-fuzzy Inference System (ANFIS) to predict change-proneness and compared it against other approaches such as Bagging, Random Forest and Logistic Regression. Malviya and Yadav [15] used k-means clustering and used Chi-Test to decide the cluster with goodness of fit among other clusters.

Research works that compared other machine learning algorithms for their prediction capability like our work recently gained a lot of momentum. Zhu et al. [16] also used OO metrics to predict change-proneness using multiple classification algorithms such as Naive Bayes, C4.5, k -NN, SVM, and an associative classification method. Malhotra and Khanna [17] investigated the effectiveness of logistic regression models against other machine learning algorithms such as Bagging, Random Forest and Multi-layer Perceptron.

Sun et al. [18] go a step forward by assessing a change proposal and the ripple effects caused by it. They used formal concept analysis to assess this effect of change and then proposed a new metric to indicate systems ability to absorb the change. Similarly, Giger et al. [19] went ahead in predicting the type of code change rather than just locating the change-prone parts of a system. While most researchers used software code for change-prediction, Han et al. [20] used design models and defined measures to predict changeability at an earlier stage of software development.

It can be seen from literature that the use of object-oriented metrics to predict change-proneness in open source software is a very active area of research. In this paper, we plan to measure the capability of certain machine learning algorithms, to predict change proneness in open source software using object-oriented metrics. Nevertheless, our research work is different from others in many dimensions:

1. We used a complete version as the training set and then used it over the subsequent version as the testing set to predict the accuracy of the considered algorithms whereas others simply use a single version to build the prediction model [13]-[17].
2. We used classes of machine learning algorithms rather than using a random set of algorithms for comparison.
3. We used baseline prediction models (ZeroR and OneR) to benchmark the evaluation criteria when comparing multiple algorithms.

III. EXPERIMENT SETUP

A. Experiment Subject - Heretrix

We are focusing our research efforts on one particular open-source project, the Heritrix Project [22]. Heritrix is an open-source WebCrawler project started by the Internet Archive in 2003. The software is open source to encourage collaboration and joint development across institutions with similar needs. The Heritrix project almost matches the description of open-source projects: it includes a complete

history of code base, public mailing lists for open discussion, a web site with documentation, and provides release notes for bug tracking. Table 1 provides information regarding the number of releases of the project, the total number of classes and the percentage of classes that changed when compared to its consecutive release. We limited the releases considered in this work until version 2.

TABLE I. CHANGE DISTRIBUTION IN THE RELEASES OF HERITRIX

Version Number	Total No. of Classes	% changed
0.2.0	120	60.83%
0.4.0	164	58.54%
0.6.0	201	35.82%
0.8.0	223	60.54%
0.10.0	246	30.49%
1.0.0	263	45.25%
1.2.0	301	56.81%
1.4.0	369	59.62%
1.6.0	411	17.52%
1.8.0	417	29.50%

B. Object-Oriented Metrics

In this subsection, we present the seventeen metrics that we used to construct the prediction model. Chidamber and Kemerer [26] proposed six of these metrics. We also included some well-known size metrics and number of dependency metrics available from [2]. The definitions of these metrics are shown in Table 2, Table 3, Table 4, Table 5 and Table 6.

TABLE II. SIZE METRICS

Metrics	Description
SLOC	Source lines of code – nonempty and non-comment
NOA	Number of attributes for a class
NOM	Number of methods for a class
NPM	Number of public methods for a class
WMC	Weighted methods per class - sum of the complexities of class's methods.

TABLE III. COHESION METRICS

Metrics	Description
LCOM	Lack of Cohesion in Methods. It counts the sets of methods in a class that are not related through the sharing of some of the class's fields

TABLE IV. INHERITANCE METRICS

Metrics	Description
DIT	Depth of Inheritance – inheritance level from the object hierarchy top
NOC	Number of children – number of immediate descendants of a class

TABLE V. COUPLING METRICS

Metrics	Description
CBO	Coupling between Object Classes – number of classes coupled to a class – can occur through inheritance, function call, return and exceptions.
RFC	Response for a Class - number of different methods that can be executed when an object of that class receives a message
CA	Afferent Coupling - how many other classes use the specific class

TABLE VI. DEPENDENCY METRICS

Metrics	Description
IIP	Inbound Intra-Package Dependencies - number of classes within the same package that depend on this class
IEP	Inbound Extra-Package Dependencies - number of classes in other packages that depend on this class
OIP	Outbound Intra-Package Dependencies Afferent Coupling - number of classes of the same package that this class depends on
OEP	Outbound Extra-Package Dependencies - number of classes of other packages that this class depends on
IIPM	Inbound Intra-Package Method Dependencies - number of methods and fields in other classes of the same package that depend on this class
IEPM	Inbound Extra-Package Method Dependencies - number of methods in other packages that depend on this class

C. Machine Learning Algorithms

In this subsection, we present the six machine learning algorithms that we used to determine the change proneness in the open source software. Of the machine learning algorithms, we used two of them to establish baseline accuracy: ZeroR and OneR algorithms. We selected one machine learning algorithms from four different classes such as Probability-based, Function-based, Instance-based, and Tree-based algorithms.

Baseline Algorithms

- **ZeroR algorithm [28]** is a simple algorithm useful for getting base line performance, in our case accuracy. It ignores all predictors and relies on the target. We used this algorithm to establish baseline accuracy.
- **OneR [28]** creates a rule for each predictor in the data. It then selects the rule with the smallest total error as its one single rule. It constructs a frequency table for each predictor against the target to create a rule for a predictor. We used this algorithm to establish baseline accuracy.

NaiveBayes (Probability-based ML Algorithm)

NaiveBayes algorithm [29] is a probability-based algorithm. It requires only small amount of training set to estimate the variables necessary for explanation. It assumes that the presence or absence of a certain feature is unrelated to the presence or absence of other features. It should be stated that NaiveBayes is based on Bayes theorem.

 ISBN: 978-1-61208-367-4

Multilayer perceptron (Function-based ML Algorithm)

Multilayer perceptron [30] is a function-based algorithm. It maps input data sets into appropriate output data set. It is made up of multiple layer nodes in a directed graph that are fully connected to each other. The network allows signals to travel from the input to the output setting the weights as they propagate through. These weights are tuned for each iteration reducing the overall error for the training set.

Nearest Neighbor (Instance-based ML Algorithm)

Nearest Neighbor (Ibk) [31] is an Instance-based algorithm also known as lazy learning algorithm. It does not do any actual training or learning at first. It populate a sample of the search space with instances whose class is known. When an instance whose class is unknown is presented for evaluation, the algorithm computes its k closest neighbors, and the class is assigned by voting among those neighbors. To prevent ties we use an odd number of k.

J48 (Tree-based ML Algorithm)

J48 is a tree-based algorithm [32] is a class of algorithm that generates a pruned or unpruned C4.5 decision tree and using a divide and conquers strategy to growing the decision tree for each instance. A new unseen instance then traverses the tree until a proper classification is reached.

IV. EXPERIMENT DESIGN

This section describes the design of the experiment. In here, we define the goal of the experiment, the dependent variables and the independent variable and how they were calculated and the tools used in the experiment. This section also gives the procedure of how the experiment was carried out.

A. Goal of the Experiment

- Object of Study: Identify and Characterize change-prone classes
- Purpose: Investigate the correlation between change proneness of a class and the set of structural metrics used in this experiment
- Perspective: From the viewpoint of the researcher and practitioner
- Context: The experiment is conducted with open-source projects and certain measurement tools that are used to calculate the metrics

B. Experimental Variables

The dependent variables in this study are a Boolean variable (Changed) that indicates whether a class changed from one version to another. Any change made to a class during the evolution of a new version from a previous version reflects the change-proneness of that class.

The independent variables are the metrics used to measure the structural properties of the classes. These metrics are presented in section 3-B of this paper.

C. Experiment Hypotheses

Our major objective is to test whether we can predict future changes to a class based on a set of structural metrics. We want to demonstrate that the machine learning algorithms were able to predict change-proneness when compared to baseline algorithms.

We tested the following hypotheses on the case study:

H1: Probability-based algorithms perform better, in terms of Accuracy, Specificity. Sensitivity and Error, than baseline algorithms.

H2: Function-based algorithms perform better, in terms of Accuracy, Specificity. Sensitivity and Error, than baseline algorithms.

H3: Instance-based algorithms perform better, in terms of Accuracy, Specificity. Sensitivity and Error, than baseline algorithms.

H4: Tree-based algorithms perform better, in terms of Accuracy, Specificity. Sensitivity and Error, than baseline algorithms.

We tested the above-mentioned hypotheses by analyzing the relationship between structural metrics of a class from an early version of the system and whether any change occurred to the class during the transition from the early version to a later version.

D. Tools

In order to collect the change data from the system, we used the Beyond Compare 2 [27] tool as a code comparison tool. This tool provided us with information as to whether the code changed from one version to another. Apart from this, we used three measurement tools to obtain the OO metrics. These tools are ckjm [23], Dependency Finder [24] and the metrics 1.3.6 plug-in available for Eclipse IDE [25].

We then used Weka [33] for application of the different machine learning algorithms. Weka is Java-based tool and runs on any platform. The algorithms can either be applied directly to a dataset or called from your own Java code. We applied the stated algorithms in Section 3-C on the Heritrix [22] project metric data collected as the test subject.

E. Experiment Procedure

In our study, we used an OSS to get the classes and run different algorithms through them to get the number of changes in the code compared to different versions of the same class. We uploaded each version of the metrics from Heritrix to Weka and performed different types of algorithms with default settings for each algorithm provided by the tool. We took the number of changes most of the classes were affected with. The steps performed are as follows:

1. **Step1:** All the classes in version n are compared with the corresponding classes in version n+1 to detect changes. This detection is done using a class comparison tool. Based on this information, we populate the Class-change matrix with YES's and NO's depending on whether the class changed from the previous release or not.
2. **Step 2:** All the classes in a version for all the versions are used as an input to a metric calculation tool to calculate all the metrics used as independent

 ISBN: 978-1-61208-367-4

variable in the project. The results from this are then used to populate the Class-metric matrix with appropriate values.

3. S**tep 3:** The input and output is then imported in the data-mining tool Weka for application of the chosen machine learning algorithms. A complete version is used as a training set and the subsequent version as the testing set. This process is repeated for all the versions.
4. **Step 4:** The results of the algorithms in terms of accuracy, root-mean squared errors, sensitivity, and specificity are recorded and compared as shown in the next section.

V. RESULTS

In this work, we collected four result values and used them for comparing the various machine learning algorithms.

A. Accuracy

Accuracy is the percentage of how accurate the algorithm is in predicting the change-proneness of a class based on the OO metrics input. The accuracy of all the selected algorithms across all versions and their average accuracy is shown in Table 7.

TABLE VII. ACCURACY OF THE MACHINE LEARNING APPROACHES

	ACCURACY					
	ZeroR	**OneR**	**Naïve Bayes**	**MLP**	**IBk**	**j48**
v0.2-v0.4	58.5	69.5	80.0	71.3	64.6	71.3
v0.4-v0.6	35.8	60.7	69.7	72.6	67.2	67.2
v0.6-v0.8	39.5	62.3	57.4	59.2	62.3	62.0
v0.8-v0.10	30.5	57.0	74.0	63.0	61.0	57.3
v0.10-v1.0	54.8	63.5	64.3	66.9	70.3	68.4
v1.0-v1.2	43.2	62.1	57.8	58.5	61.8	66.1
v1.2-v1.4	59.6	62.0	53.1	62.0	61.8	62.6
v1.4-v1.6	17.5	52.6	77.6	59.1	55.5	57.9
v1.6-v1.8	70.5	74.0	75.5	74.6	76.0	75.3
	45.5	62.6	67.7	65.2	64.5	65.3

a. Values are percentages

B. Specificity

Specificity is the percentage of the values that were originally “No” and also predicted as “No” as obtained from the confusion matrix. After applying the algorithms, the result was that the baseline specificity has a fair specificity better than the other algorithms specificity. ZeroR algorithm was the best giving perfect specificity for five times, as shown in Table 8.

TABLE VIII. SPECIFICITY OF THE MACHINE LEARNING APPROACHES

	SPECIFICITY					
	ZeroR	**OneR**	**Naïve Bayes**	**MLP**	**IBk**	**j48**
v0.2-v0.4	100	54.4	33.8	44.1	39.7	33.8
v0.4-v0.6	100	44.2	18.6	24.8	43.4	39.5
v0.6-v0.8	0	13.6	4.5	14.8	9.1	14.8
v0.8-v0.10	100	56.1	17.5	44.4	45.6	50.9
v0.10-v1.0	0	11.8	7.6	7.6	11.8	9.7
v1.0-v1.2	0	16.9	10.0	10.8	24.6	20.0
v1.2-v1.4	100	47.0	13.4	37.6	36.2	38.9
v1.4-v1.6	100	54.9	15.6	43.4	49.6	45.7
v1.6-v1.8	0	2.0	5.1	1.7	7.5	3.1
	55.6	33.4	14.0	25.5	29.7	28.5

a. Values are percentages

C. Sensitivity

Sensitivity is the percentage of values, which were “yes” and also predicted as “yes” as obtained from the confusion matrix. After applying the algorithms, the result was that the baseline sensitivity has a fair sensitivity better than the other algorithms sensitivity. ZeroR algorithm was the best giving perfect sensitivity for five times. Table 9 shows the sensitivity result of all the algorithms on the given versions.

TABLE IX. SENSITIVITY OF THE MACHINE LEARNING APPROACHES

	SENSITIVITY					
	ZeroR	**OneR**	**Naïve Bayes**	**MLP**	**IBk**	**j48**
v0.2-v0.4	100	86.5	89.6	82.3	67.7	75.0
v0.4-v0.6	100	69.4	48.6	68.1	86.1	79.2
v0.6-v0.8	0	46.7	32.6	42.2	43.7	46.7
v0.8-v0.10	100	86.7	54.7	80.0	76.0	76.0
v0.10-v1.0	0	33.6	30.3	36.1	48.7	42.0
v1.0-v1.2	0	46.2	33.3	35.1	51.5	55.6
v1.2-v1.4	100	68.2	30.5	61.8	60.5	63.6
v1.4-v1.6	100	87.5	45.8	70.8	79.2	75.0
v1.6-v1.8	0	16.3	29.3	17.9	36.6	23.6
	55.6	60.1	43.8	54.93	61.1	59.6

a. Values are percentages

D. Root Mean Squared Error (RMSE)

RMSE is the difference between values predicted by a model and the values actually observed. After applying the algorithms, the result was that the Multilayer Perceptron

 ISBN: 978-1-61208-367-4

algorithm had the lowest RMSE. Table 10 shows the RMSE result of all the algorithms on the given versions.

TABLE X. RMSE OF THE MACHINE LEARNING APPROACHES

	ROOT MEAN SQUARED ERROR					
	ZeroR	**OneR**	**Naïve Bayes**	**MLP**	**IBk**	**j48**
v0.2-v0.4	0.49	0.55	0.45	0.47	0.59	0.51
v0.4-v0.6	0.53	0.63	0.54	0.48	0.57	0.55
v0.6-v0.8	0.55	0.61	0.65	0.59	0.61	0.59
v0.8-v0.10	0.55	0.66	0.50	0.56	0.62	0.61
v0.10-v1.0	0.52	0.60	0.58	0.48	0.54	0.54
v1.0-v1.2	0.51	0.62	0.64	0.51	0.61	0.53
v1.2-v1.4	0.49	0.62	0.68	0.48	0.62	0.53
v1.4-v1.6	0.57	0.69	0.46	0.55	0.66	0.61
v1.6-v1.8	0.47	0.51	0.49	0.44	0.49	0.46
	0.52	0.61	0.55	0.51	0.59	0.55

VI. DISCUSSION

In this section, we will discuss our four comparison criteria used. First, we have accuracy. After applying the algorithms, the result was that all of the machines learning algorithms have a fair accuracy better than the baseline accuracy. Naive Bayes and Nearest Neighbor are better considering both have given the comparatively better results than the most and Naïve Bayes gave the highest percentage with 68% as shown in Table 7. Then, there is specificity. After applying the algorithms, the result was that the baseline specificity has a fair specificity better than the other algorithm's specificity. ZeroR algorithm was the best giving perfect specificity for five times as shown in Table 8. Next is Sensitivity. After applying the algorithms, the result was that the nearest neighbor algorithm was the only one with a better average sensitivity percentage compared to the baseline algorithm as shown in Table 9. Finally, we have RMSE. After applying the algorithms, the result was that the Multilayer Perceptron algorithm had the lowest RMSE, which was better than the baseline algorithm as shown in Table 10. Based on this analysis, we reject all the hypotheses H1, H2, H3 and H4 as no class of machine learning algorithm performs better than the baseline algorithms in terms of all considered comparison factors: accuracy, sensitivity, specificity and error.

VII. THREATS TO VALIDITY

This section discusses the threats to validity in this study and the way they were treated throughout the experiment.

A. Construct Validity

Construct Validity is the degree to which the independent variables and dependent variables accurately measure the concepts they purport to measure. The dependent variables we used in our study were change, which is a Boolean variable as to whether the class changed or not. As the way they are computed is straightforward, we consider them constructively valid. If any, the way the size was calculated can pose a slight threat if there is a better way for it.

B. External Validity

External Validity is the degree to which the results of the research can be generalized to the population under study and other research settings. A crucial threat lies with the size of the case study considered. Only a single project releases are considered in this study with 10 releases. This may affect the generalization of the identified conclusions. On the same lines, another valid threat that cannot be excluded until extensive empirical results are collected is that the case study will reflect the characteristics from a specific domain. In addition, the data collected from the open source project was by analyzing the code. Poor documentation can affect the results of the analysis significantly.

C. Internal Validity

Internal Validity is the degree to which conclusions can be drawn about the casual effect of independent variables on the dependent variables. Apart from the variables considered, our approach might have omitted other important variables that can serve as predictors. In addition, the size of the open source project can be considered as a potential threat as our project was not very big, but significantly large.

VIII. CONCLUSION AND FUTUREWORK

This paper reported findings of an empirical study conducted to investigate the measures that affect the change-proneness of classes in an open source project. The goal was to use a case study from the open source community in order to explore the relationship between the structural characteristics of the project and the change proneness of classes within that project from one version to the other. The study concluded a lot of interesting results that conform to previous studies, such as size-related metrics and coupling metrics are correlated with change proneness. In conclusion to the experiment, we believe that using machine learning algorithms to predict change proneness in open source software using object-oriented metrics is an excellent field for research and needs to be further investigated. In many cases, we were able to identify that the baseline accuracy performed better than the machine learning algorithms considered. This result calls for more research for better algorithms that can be used for prediction of change-proneness.

It should be noted that we used the default setting of Weka for all the machine-learning algorithms used. Moreover, as a future step, we plan to change the settings of certain parameters in these algorithms to find a better prediction model for software change proneness. In addition, we plan to make use of correlation and principal component analysis to select only those metrics that seem to affect the change-proneness. In addition, the study provided some useful information regarding dependency relationships and their association. Based on our findings, we suggest

 ISBN: 978-1-61208-367-4

practitioners dealing with open source projects to collect static metrics and change data as part of their development effort. This data can be used to prioritize preventive action on the classes that are still under development.

ACKNOWLEDGMENT

The authors acknowledge the support of King Faisal University in the development of this work.

REFERENCES

[1] M. M. Lehman and L. A. Belady, Program evolution: processes of software change. CA: Academic Press Professional, Inc., 1985.

[2] A. Güneş Koru and H. Liu, "Identifying and characterizing change-prone classes in two large-scale open-source products," Journal of Systems and Software, vol. 80, Jan. 2007, pp. 63-73, doi: 10.1016/j.jss.2006.05.017.

[3] E. S. Raymond, The Cathedral & the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary. CA: O'Reilly Media, Inc., 2001.

[4] King Abdulaziz City for Science and Technology, National Program for Free & Open Source Software Technology. Available: http://www.motah.org.sa. Retrieved: August, 2014.

[5] C. Årdal, A. Alstadsæter, and J.-A. Røttingen, "Common characteristics of open source software development and applicability for drug discovery: a systematic review," Health Research Policy and Systems, vol. 9, Sept. 2011, pp. 1-16, doi:10.1186/1478-4505-9-36.

[6] T. Gyimothy, R. Ferenc, and I. Siket, "Empirical Validation of Object-Oriented Metrics on Open Source Software for Fault Prediction," IEEE Trans. Softw. Eng., vol. 31, Oct. 2005, pp. 897-910, doi: 10.1109/tse.2005.112.

[7] C. van Koten and A. R. Gray, "An application of Bayesian network for predicting object-oriented software maintainability," Inf. Softw. Technol., vol. 48, Jan. 2006, pp. 59-67, doi: 10.1016/j.infsof.2005.03.002.

[8] Y. Zhou and H. Leung, "Predicting object-oriented software maintainability using multivariate adaptive regression splines," Journal of Systems and Software, vol. 80, Aug. 2007, pp. 1349-1361, doi: 10.1016/j.jss.2006.10.049.

[9] S. Eski and F. Buzluca, "An Empirical Study on Object-Oriented Metrics and Software Evolution in Order to Reduce Testing Costs by Predicting Change-Prone Classes," Proc. IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops (ICSTW 11), IEEE, Mar 2011, pp. 566 – 571, doi: 10.1109/icstw.2011.43.

[10] F. Khomh, M. Penta, Y.-G. Guéhéneuc, and G. Antoniol, "An exploratory study of the impact of antipatterns on class change- and fault-proneness," Empirical Software Engineering, vol. 17, June 2012, pp. 243-275,doi: 10.1007/s10664-011-9171-y.

[11] D. Romano, P. Raila, M. Pinzger, and F. Khomh, "Analyzing the Impact of Antipatterns on Change-Proneness Using Fine-Grained Source Code Changes," Proc.19th Working Conference on Reverse Engineering (WCRE 12), IEEE Computer Society,Oct. 2012, pp. 437-446, doi: 10.1109/WCRE.2012.53.

[12] H. Lu, Y. Zhou, B. Xu, H. Leung, and L. Chen, "The ability of object-oriented metrics to predict change-proneness: a meta-analysis," Empirical Software Engineering, vol. 17, June 2012, pp. 200-242,doi: 10.1007/s10664-011-9170-z.

[13] M. O. Elish and M. Al-Rahman Al-Khiaty, "A suite of metrics for quantifying historical changes to predict future change-prone classes in object-oriented software," J. Softw. Evol. and Proc., vol. 25, May 2013, pp. 407-437,doi: 0.1002/smr.1549.

[14] A. Peer and R. Malhotra, "Application of adaptive neuro-fuzzy inference system for predicting software change proneness," Proc. Advances in Computing, Communications and Informatics (ICACCI 13), IEEE, Aug. 2013, pp. 2026-2031, doi: 10.1109/ICACCI.2013.6637493.

[15] A. K. Malviya and V. K. Yadav, "Maintenance activities in object oriented software systems using K-means clustering technique: A review," Proc. CSI Sixth International Conference on Software Engineering (CONSEG 12), IEEE, Sept. 2012, pp. 1-5, doi: 10.1109/CONSEG.2012.6349490.

[16] X. Zhu, Q. Song, and Z. Sun, "Automated Identification of Change-Prone Classes in Open Source Software Projects," Journal of Software, vol. 8, Feb. 2013, pp. 361-366, doi: 10.4304/jsw.8.2.361-366.

[17] R. Malhotra and M. Khanna, "Investigation of relationship between object-oriented metrics and change proneness," International Journal of Machine Learning and Cybernetics, vol. 4, Aug. 2013, pp. 273-286, doi: 10.1007/s13042-012-0095-7.

[18] X. Sun, B. Li, and Q. Zhang, "A Change Proposal Driven Approach for Changeability Assessment Using FCA-Based Impact Analysis," Proc. IEEE 36th Annual Computer Software and Applications Conference (COMPSAC 12), IEEE, Jul. 2012, pp. 328 – 333, doi: 10.1109/COMPSAC.2012.44.

[19] E. Giger, M. Pinzger, and H. C. Gall, "Can we predict types of code changes? An empirical analysis," Proc. 9th IEEE Working Conference on Mining Software Repositories (MSR 12), IEEE, Jun. 2012, pp. 217-226, doi: 10.1109/MSR.2012.6224284.

[20] A.-R. Han, S.-U. Jeon, D.-H. Bae, and J.-E. Hong, "Measuring behavioral dependency for improving change-proneness prediction in UML-based design models," Journal of Systems and Software, vol. 83, Feb. 2010, pp. 222-234, doi: 10.1016/j.jss.2009.09.038.

[21] E. Arisholm and L. C. Briand, "Predicting fault-prone components in a java legacy system," Proc. ACM/IEEE International Symposium on Empirical Software Engineering (ISESE 06), ACM, 2006, pp. 8-17, doi: 10.1145/1159733.1159738.

[22] P. Jack. Heritrix. Available: https://webarchive.jira.com/wiki/display/Heritrix/Heritrix, Retrieved: Sept. 2014.

[23] D. D. Spinellis. (2008). ckjm — Chidamber and Kemerer Java Metrics. Available: http://www.spinellis.gr/sw/ckjm/, Retrieved: Sept. 2014.

[24] J. Tessier. Dependency Finder. Available: http://depfind.sourceforge.net/, Retrieved: Sept. 2014.

[25] F. Sauer, Metrics 1.3.6. Available: http://metrics.sourceforge.net/, Retrieved: Sept. 2014.

[26] S. R. Chidamber and C. F. Kemerer, "A Metrics Suite for Object Oriented Design," IEEE Trans. Softw. Eng., vol. 20, Jun. 1994, pp. 476-493, doi: 10.1109/32.295895.

[27] Scooter Software. Beyond Compare 2. Available: http://www.scootersoftware.com/, Retrieved: Sept. 2014.

[28] I. H. Witten, E. Frank, M. A. Hall, Data Mining: Practical Machine Learning Tools and Techniques, MA: Morgan Kauffman Publishers, 2011.

[29] D. Heckerman, D. Geiger, D. M. Chickering, "Learning Bayesian networks: The combination of knowledge and statistical data," Machine Learning, vol. 20(3), Sept. 1995, pp. 197–243, doi: 10.1007/BF00994016.

[30] C. M. Bishop, Neural networks for pattern recognition, New York: Oxford University Press, 1995.

[31] D. Aha, "Tolerating noisy, irrelevant, and novel attributes in instance-based learning algorithms," International Journal of Man-Machine Studies, vol. 36(2), Feb. 1992, pp. 267–287, doi: 10.1016/0020-7373(92)90018-G.

[32] J. Quinlan, C4.5: Programs for Machine Learning. CA: Morgan Kaufmann, 1993.

[33] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA Data Mining Software: An Update," SIGKDD Explorations, vol. 11, Jun. 2009, pp. 10-18, doi: 0.1145/1656274.1656278.

 ISBN: 978-1-61208-367-4