Fault-Proneness of Open Source Systems: An Empirical Analysis

Mamdouh Alenezi
College of Computer & Information Sciences, Prince Sultan University, Riyadh 11586, Saudi Arabia
Shadi Banitaan
College of Engineering & Science University of Detroit Mercy Detroit, MI 48221, USA
Qasem Obeidat
Department of Software Engineering, Al-Zaytoonah University of Jordan, Amman 11377, Jordan

Abstract: *Developing quality software is a very complex job considering the complexity and size of software developed these days. Early prediction of software quality assists in optimizing testing resources. Many fault prediction models have been developed using several internal attributes and different machine learning techniques. However, the open-source community still lacks a concise knowledge about what types of internal attributes affect the software quality the most. In this work, an empirical investigation is conducted to explore the relationships between internal attributes of open-source systems and their fault-proneness. The results of the empirical analysis showed that by selecting only nine internal attributes, the fault prediction models accuracy did not decrease significantly. This indicates that only a subset of these internal attributes is worth collection and investigation. By focusing on a small set of internal attributes, the quality assurance team can save time and resources while achieving high accuracy fault-proneness predictions.*

Keywords: *fault-proneness, open-source systems, internal attributes, quality.*

1. Introduction

Even when applying the principles of the software development methodologies, a fault-free software system is very difficult to achieve. The maintenance phase of software projects is a very challenging and costly. The extent of resources spent on software maintenance is much more than what is being spent on its development. Consequently, any part of maintenance that can be automated will eventually lead to saving maintenance resources. One possible area where an effort is beneficial to lower the maintenance costs is identifying the source code parts that are presumably to encompass faults and therefore require changes.

A noteworthy research work has been devoted to describing particular quality metrics and building quality predictive models based on internal attributes. Fault prediction models are usually built using software metrics and previously collected fault data. These models are then utilized to guide decision- making in the course of development. Fault-proneness is the most frequently investigated dependent variable [1]. Predicting the classes fault-proneness helps in focusing the effort of validation and verification, which helps in finding more faults for the same effort. In case of predicting a class is as likely to be faulty, corrective actions can be invested to test and inspect the class. Fault prediction will channel the focus of the developers to carefully examine and test the files or classes that have a high probability of defectiveness. Focusing the effort on faulty classes will help in managing and utilizing the resources of the software project more efficiently. This will make the maintenance phase easier for both the customers and the project owners.

Software fault prediction models depend on the information available in software metrics. The software metrics data quality plays a significant role in building accurate prediction models. The selection of a subset of the software metrics is an essential part of the model building process. Focusing on a subset of these metrics will save the time needed to collect and manage them. In addition, using a reduced metrics set in building predictive models will lead to better classification speed. In this study, we investigate the association between internal quality attributes (source code metrics) and fault-proneness of open source projects.

The remainder of this paper is organized as follows. Section 2 presents the used methodology. The experimental evaluation is given in Section 3. Some threats to validity are presented in Section 4. Section 5 discusses related work. Conclusions of the research are presented in Section 6.

2. Methodology

To select a subset of software metrics that are sufficient to predict faulty classes, this study investigate twenty internal attributes of eight open source software systems. This study compares four different classifications. It also applies a feature selection technique to find the subset of metrics that are sufficient to predict faulty classes in open-source software projects.

2.1 Feature Selection

Hall and Holmes [2] categorized feature selection algorithms to (1) algorithms that evaluate individual attributes and (2) algorithms that evaluate subset of attributes. The first category of feature selection algorithms identifies which metric is able to serve as discriminatory attribute for indicating an external quality. The second category selects a subset of features that are best to identify the class label. In this study, the second category is selected since the goal is to identify which subsets of metrics are better in identifying faulty classes in open source software systems.

Feature ranking algorithms evaluate attributes individually based on a certain measure and order them accordingly. Although that some attributes may be less useful by themselves but the can make a substantial contribution when combined with other attributes. Feature subset selection

methods handle that by selecting and searching for subsets of attributes that collectively have good performance. We utilized correlation- based feature selection (CFS) technique to find important metrics. CFS uses a search algorithm along with a function to estimate the worth of feature subsets, similar to the most of feature selection algorithms. CFS measures the individual predictive ability of each feature to estimate the value of a subset of attributes taking into account the redundancy between them. Based on a previous study [3] higher performance was achieved using correlation-based feature selection.

2.2 Classification Algorithms

Several modeling techniques are available to build fault-proneness models like regression and classification techniques. Classification is one of the most commonly used machine learning techniques. It is also known as supervised statistical learning. In supervised learning, the model needs to be first trained using data with predetermined classes. This data is used to train the learning algorithm, which creates a model that can then be used to label/classify the testing instances, where the values of the class labels are unknown. We compare four different classifiers namely Naive Bayes (NB), Bayesian Networks (BNet), J48, and Random Forests (RF). These classification algorithms are known to be high-performance fault predictors [1]. The WEKA default settings of these algorithms were used in this study [4].

2.3 Data Collection

The data of defects used in this study was gathered by Jureczko and Spinellis [5] and is available online at the PROMISE repository. For that study, the defects were collected using BugInfo1 tool, from the selected software systems source code repositories. BugInfo analyzes the logs of the code repositories and decides if a commit is a bug fix or not based on the log content. These data were widely used to construct fault-proneness predictive models in the literature [6], [7], [8].

2.4 The Internal Attributes

The internal attributes of software namely coupling, cohesion, inheritance and size are the independent variables used in this study. The metrics come from several metrics suites. We focus on object-oriented metrics to be independent variables in a prediction model, which is accessible at early stages of software development. The selected internal attributes (metrics) of open source software systems are shown in more detail in Table I.

2.5 Subject Software Systems

To select the systems for the empirical analysis, four selection criteria have been used. First, the selected systems had to be well-known systems that are very widely used. Second, the systems had to be sizable, so the systems can be realistic and have multi-developers. Third, the systems had to be actively maintained. Finally, the data of these systems had to be publicly available. Eight various-sized systems have been chosen from different domains. Characteristics of the selected software systems are listed in Table II.

TABLE II. SELECTED SOFTWARE SYSTEMS

System	Ver	Classes
Camel	1.6	965
Ant	1.7	745
Xerces	1.4.4	587
jEdit	4.3	493
POI	3.0	442
Ivy	2.0	352
Lucene	2.4	340
Synapse	1.2	256

3. Experimental Evaluation

In this experimental evaluation, several machine learning algorithms are investigated. The goal is to see which one of them achieves better results in the case of open source systems. Then, we show the results of the feature selection study.

3.1 Classification Metrics

To build and evaluate the prediction models, 10-fold cross validation has been performed to obtain unbiased evaluation results. Comparisons between classifiers are based on two measures namely AUC and F-measure. AUC is the area under the receiver operating characteristics curve whereas F-measure is the harmonic mean of Precision and Recall. AUC is the most informative indicator of predictive accuracy within the field of software defect prediction [9].

3.2 Classification Results

To investigate which classifier lead to best fault prediction performance, prediction models are built using all the internal attributes (twenty software metrics). Table III shows the F- measure and AUC values for each project. Using the F- measure, both BNet and RF achieve better results compared to the other two classifiers. As stated earlier, the analysis primarily focuses on the AUC value for the different classifiers. The results show that Xerces has the highest AUC value whereas jEdit has the smallest AUC value. The AUC values found in the experiments are relatively high. We found that in most cases, the RF classifier is the best performing technique based on AUC while the J48 classifier is the worst. The median value of the RF classifier is 0.78 which is the highest.

TABLE III. CLASSIFICATION RESULTS

System	AUC				F-measure			
	NB	BNet	J48	RF	NB	BNet	J48	RF
Camel	0.675	0.638	0.616	0.722	0.766	0.746	0.753	0.775
Ant	0.806	0.81	0.665	0.807	0.809	0.782	0.792	0.806
Xerces	0.845	0.922	0.913	0.941	0.677	0.825	0.938	0.927
jEdit	0.594	0.447	0.467	0.72	0.95	0.967	0.967	0.966
POI	0.805	0.879	0.772	0.872	0.524	0.812	0.779	0.781
Ivy	0.765	0.79	0.681	0.736	0.852	0.842	0.855	0.857
Lucene	0.728	0.697	0.687	0.757	0.583	0.633	0.678	0.695
Synapse	0.756	0.744	0.692	0.796	0.743	0.747	0.741	0.738
Median	0.761	0.767	0.684	0.777	0.755	0.797	0.786	0.794

TABLE I. METRICS DEFINITIONS

Metric Name	Definition
Weighted methods per class (WMC)	Based on the assumption of unity weights of methods, WMC is the number of methods in the class.
Depth of Inheritance Tree (DIT)	The DIT metric provides for each class a measure of the inheritance levels from the object hierarchy top.
Number of Children (NOC)	The NOC metric measures the number of immediate descendants of the class.
Coupling between object classes (CBO)	The CBO metric represents the number of classes coupled to a given class. This couplings can occur through method calls, field accesses, inheritance, method arguments, return types, and exceptions.
Response for a Class (RFC)	The RFC metric measures the number of different methods that can be executed when an object of that class receives a message.
Lack of cohesion in methods (LCOM)	The LCOM metric counts the sets of methods in a class that are not related through the sharing of some of the class fields.
Lack of cohesion in methods (LCOM3)	LCOM3 varies between 0 (high cohesion) and 1 (no cohesion). 0 means that each method accesses all variables (i.e., highest possible cohesion), whereas 1 indicates extreme lack of cohesion.
Afferent couplings (Ca)	The Ca metric represents the number of classes that depend upon the meas-ured class.
Efferent couplings (Ce)	The Ca metric represents the number of classes that the measured class is depended upon.
Number of Public Methods (NPM)	The NPM metric counts all the methods in a class that are declared as public.
Data Access Metric (DAM)	This metric is the ratio of the number of private (protected) attributes to the total number of attributes declared in the class.
Measure of Aggre-gation (MOA)	The metric is a count of the number of class fields whose types are user defined classes.
Measure of Functional Abstraction (MFA)	This metric is the ratio of the number of methods inherited by a class to the total number of methods accessible by the member methods of the class.
Cohesion Among Methods of Class (CAM)	This metric computes the relatedness among methods of a class based upon the parameter list of the methods.
Inheritance Coupling (IC)	This metric provides the number of parent classes to which a given class is coupled. A class is coupled to its parent class if one of its inherited methods functionally dependent on the new or redefined methods in the class.
Coupling Between Methods (CBM)	The metric measures the total number of new/redefined methods to which all the inherited methods are coupled.
Average Method Complexity (AMC)	This metric measures the average method size for each class.
McCabe's cyclo-matic complexity (CC)	CC is equal to number of different paths in a method (function) plus one.
Lines of Code (LOC)	LOC is the sum of number of fields, number of methods and number of instructions in every method of the investigated class.

3.3 Correlation Results

To understand the relationships between software metrics, their correlation coefficient (the strength of relationship among their counterparts) can be measured. Metrics that correlate with each other means that they measure similar aspects of software modules. The Spearman correlation is preferred instead of Pearson correlation because the former ignores any assumptions about the data distribution. Table IV shows the Spearman rank correlations between coupling metrics. Correlations greater than 0.69 are highlighted since they indicate strong correlation. The Table also shows that the following coupling metrics are strongly correlated with each other:

- WMC strongly correlates with RFC, LCOM, NPM, LOC, CAM.
- DIT strongly correlates with MFA. This strong cor- relation can be explained since DIT and MFA are measures of inheritance.
- RFC strongly correlates with LOC, CAM, AMC, and maxCC.
- LOC strongly correlates with WMC, RFC, and Ce.
- LCOM3 strongly correlates with DAM. This strong correlation can be explained since these measures tries to explore the cohesion of methods and attributes inside a class.
- LOC strongly correlates with AMC and maxCC. This strong correlation can be explained since AMC and maxCC are measures of complexity.
- CAM strongly correlates with WMC, RFC, NPM, and LOC.
- IC strongly correlates with CBM. This strong correlation can be explained since a class is coupled to its parent class (in case of IC) if one of its inherited methods is functionally dependent on the new or redefined methods, while CBM is the total number of new/redefined methods.
- maxCC strongly correlates with avgCC. This strong correlation can be explained since these two measures are a variation of the McCabe's cyclomatic complexity.

A. Internal Attribute Selection Results

In this section, correlation-based feature selection (CFS) was used to find germane metrics. Based on the execution of

CFS, these product metrics were selected (CBO, RFC, LCOM, Ca, Ce, LCOM3, MFA, CAM, IC). Based on the classification results shown in Table III, the Random Forrest classifier achieved the best median value with comparison with the remainder of the classifiers. To evaluate the new predictive model that contains only a subset of the product metrics, Random Forest with 10-fold cross validation is performed. The results of the prediction accuracy are shown in Table V.

TABLE V. RANDOM FORREST CLASSIFICATION RESULTS WITH FEATURE SELECTION

System	AUC	F-measure
Camel	0.68	0.77
Ant	0.8	0.786
Xerces	0.937	0.924
jEdit	0.967	0.714
POI	0.869	0.781
Ivy	0.741	0.857
Lucene	0.746	0.682
Synapse	0.801	0.73

TABLE VI. THE RESULTS OF TWO KRUSKAL-WALLIS TESTS FOR AUC AND F-MEASURE

	Test 1 AUC	**Test 2 F-measure**
chi-squared	7	7
degree of freedom	7	7
p-value	0.429	0.429

To investigate whether the difference between using all metrics data or the reduced metrics data in building predictive models is significant or not, the Kruskal-Wallis test is executed. The Kruskal-Wallis test [10] is a nonparametric alternative to the one-way analysis of variance (ANOVA). The Kruskal- Wallis's test fits this case because it has eight independent samples of AUC and F-measure values for each software project. The Kruskal-Wallis's test was executed individually for AUC and F-measure values (see Table VI). In both tests, the p-value is large, which indicates that the difference between the AUC values is statistically insignificant. In other words, we can use the reduced data set (9 metrics) instead of using all data set (20 metrics) in building predictive models. This result shows that by collecting only these product metrics (CBO, RFC, LCOM, Ca, Ce, LCOM3, MFA, CAM, IC), the resulting predictive model accuracy does not decrease significantly if you collect all the studied metrics. This shows that only these selected metrics are important in distinguishing between faulty and non-faulty classes.

4. Threats to Validity

In this section, we identify and discuss the main threats to the validity of this empirical study. To mitigate conclusion validity, we performed the experiments 10 times to achieve reliable results. Tests were performed using 10-fold cross validation. Two measures were used to evaluate the performance of different classifiers namely the Area Under Receiver Operating Characteristics Curve (AUC) and F-measure. To mitigate construct validity, previously validated and well-known software metrics were used. The used metrics were satisfactory and widely used in software fault prediction.

5. Related Work

This section reviews the studies that discussed internal attribute and fault proneness of open source systems. Briand et al. [11] conducted a case study that investigated the relationships between quality factors of object-oriented design and fault-proneness of an open source system (LALO). They found a number of metrics from the Chidamber and Kemerer (CK) metrics suite were statistically associated with the fault- proneness of classes. In a later study, they replicated the study [12], and found different results, including the relations between DIT, NOC and fault-proneness of classes.

Gyimothy et al. [13] conducted an empirical investigation of eight object-oriented metrics for fault prediction of the Mozilla project. They used several statistical methods such as linear and logistic regression, decision trees and neural net- works. Their results showed strong statistically significant correlations between most of the CK metrics and fault-proneness. Olague et al. [14] conducted an empirical investigation using three object-oriented metrics suites and the Mozilla Rhino project. Their study showed that some of metrics used in the study are consistent predictors of class fault-proneness.

Zhou and Leung [15] took into account the severity of faults. They conducted their study on a public dataset from the NASA Metrics Data Program. Their results showed a great number of strong correlations between the CK metrics and high, low and ungraded severity faults. In this work, several pre-validated metrics are used to predict the fault-proneness of eight open source systems. This study is different than these previous studies since it uses eight large open source software systems that come from different domains. The other difference is the fact that the dataset used in this study is publicly available and has been used before to construct fault prediction models.

6. CONCLUSION

This work empirically investigated to what extent and how fault-proneness could be explained by means of internal quality attributes. Software quality was measured in terms of 20 internal quality attributes. This case study showed that most of the internal attributes of open source project correlate with each other. Internal attributes that correlate with each other means that they measure similar concepts. The results also showed that selecting only a subset of these internal attributes achieved similar accuracy results compared to using all the internal attributes. This shows that certain internal attributes can be collected and investigated, which saves the project time. This allows software quality assurance teams to concentrate on a subset of internal attributes.

References

[1] C. Catal, "Software fault prediction: A literature review and current trends," *Expert Systems with Applications*, vol. 38, no. 4, pp. 4626– 4636, 2011.

[2] M. A. Hall and G. Holmes, "Benchmarking attribute selection techniques for discrete class data mining," *IEEE Transactions on Knowledge and Data Engineering*, vol. 15, no. 6, pp. 1437–1447, 2003.

[3] 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.

[4] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining software: an update," *ACM SIGKDD explorations newsletter*, vol. 11, no. 1, pp. 10–18, 2009.

[5] 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.

[6] G. Scanniello, C. Gravino, A. Marcus, and T. Menzies, "Class level fault prediction using software clustering," in *Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering (ASE2013)*. IEEE, 2013.

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

[8] M. Alenezi and K. Magel, "Empirical evaluation of a new coupling metric: Combining structural and semantic coupling," *International Journal of Computers and Applications*, vol. 36, no. 1, 2014.

[9] M. Baojun, K. Dejaeger, J. Vanthienen, and B. Baesens, "Software defect prediction based on association rule classification," *Available at SSRN http://ssrn.com/abstract=1785381, 2011*.

[10] P. Sprent and N. C. Smeeton, *Applied nonparametric statistical meth- ods*, 4th ed., ser. Chapman & Hall/CRC Texts in Statistical Science. Boca Raton, FL: CRC Press, 2007.

[11] L. C. Briand, J. Wüst, S. V. Ikonomovski, and H. Lounis, "Investigating quality factors in object-oriented designs: an industrial case study," in *Proceedings of the 21st international conference on Software engineer- ing*. ACM, 1999, pp. 345–354.

[12] L. Briand, J. Wüst, and H. Lounis, "Replicated case studies for inves- tigating quality factors in object-oriented designs," *Empirical Software Engineering: An International Journal*, vol. 6, pp. 11–58, 2001.

[13] 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.

[14] H. M. Olague, L. H. Etzkorn, S. Gholston, and S. Quattlebaum, "Empirical validation of three software metrics suites to predict fault- proneness of object-oriented classes developed using highly iterative or agile software development processes," *IEEE Transactions on Software Engineering*, vol. 33, no. 6, pp. 402–419, 2007.

[15] Y. Zhou and H. Leung, "Empirical analysis of object-oriented design metrics for predicting high and low severity faults," *IEEE Transactions on Software Engineering*, vol. 32, no. 10, pp. 771–789, 2006.

TABLE IV. SPEARMAN CORRELATION RESULTS

	WMC	DIT	NOC	CBO	RFC	LCOM	Ca	Ce	NPM	LCOM3	LOC	DAM	MOA	MFA	CAM	IC	CBM	AMC	maxCC	avgCC
WMC	1.00																			
DIT	0.11	1.00																		
NOC	0.17	-0.01	1.00																	
CBO	0.46	0.05	0.24	1.00																
RFC	**0.83**	0.20	0.13	0.57	1.00															
LCOM	**0.77**	0.09	0.15	0.37	0.60	1.00														
Ca	0.23	-0.17	0.31	0.62	0.14	0.18	1.00													
Ce	0.46	0.25	0.09	0.69	0.67	0.36	0.10	1.00												
NPM	**0.89**	0.09	0.12	0.34	0.65	0.69	0.19	0.31	1.00											
LCOM3	-0.31	-0.10	-0.08	-0.17	-0.38	0.09	-0.05	-0.24	-0.25	1.00										
LOC	**0.76**	0.18	0.13	0.51	**0.93**	0.49	0.15	0.60	0.57	-0.42	1.00									
DAM	0.42	0.16	0.14	0.27	0.48	0.12	0.05	0.35	0.34	**-0.78**	0.48	1.00								
MOA	0.43	0.07	0.11	0.39	0.45	0.22	0.21	0.44	0.32	-0.34	0.46	0.40	1.00							
MFA	-0.04	**0.93**	-0.02	-0.01	0.08	-0.02	-0.20	0.20	-0.07	-0.05	0.06	0.08	-0.01	1.00						
CAM	**-0.78**	-0.07	-0.12	-0.45	**-0.72**	-0.58	-0.22	-0.49	-0.65	0.28	-0.69	-0.36	-0.43	0.07	1.00					
IC	0.28	0.63	-0.01	0.14	0.32	0.18	-0.07	0.29	0.21	-0.16	0.31	0.17	0.13	0.58	-0.23	1.00				
CBM	0.30	0.61	-0.01	0.13	0.32	0.21	-0.06	0.29	0.24	-0.15	0.31	0.16	0.12	0.56	-0.25	**0.97**	1.00			
AMC	0.35	0.17	0.04	0.38	**0.71**	0.14	0.01	0.52	0.15	-0.40	**0.84**	0.38	0.31	0.15	-0.38	0.24	0.23	1.00		
maxCC	0.54	0.05	0.09	0.43	**0.71**	0.36	0.16	0.48	0.39	-0.32	**0.76**	0.35	0.33	-0.04	-0.47	0.21	0.20	0.69	1.00	
avgCC	0.44	-0.09	0.05	0.36	0.57	0.30	0.18	0.34	0.33	-0.16	0.59	0.21	0.22	-0.18	-0.35	0.09	0.09	0.53	**0.89**	1.00