Predicting Object-Oriented Class Fault-Proneness: A Replication Study

Jehad Al Dallal*
Department of Information Science, Kuwait University, P.O. Box 5969, Safat 13060, Kuwait.

* Corresponding author. Email: j.aldallal@ku.edu.kw
Manuscript submitted December 5, 2017; accepted May 15, 2018.
doi: 10.17706/jsw.13.5.269-276

Abstract: Fault-proneness is a software quality attribute. It refers to the extent to which a software module is prone to faults. In object-oriented software development, the class is the basic design unit, and its quality affects the software's overall quality. Fault-proneness cannot be measured during early software development phases before the faults are detected and fixed. Several studies empirically explored the ability of statistical-based models that use other software artifacts known at early software development phases to predict fault-proneness. In this paper, we report a replication study that empirically investigated models' ability based on six well-known and commonly applied design quality measures to predict class fault-proneness using five Java open-source systems. The results indicated that in most cases, the models based on the considered quality measures were found to be statistically significant class fault-proneness predictors. In addition, considering the measures in combination allows for building prediction models with acceptable classification performance.

Key words: Fault-proneness, object-oriented class, quality attribute, quality measure.

1. Introduction

Software quality is an important factor considered in software engineering. During software development stages, software engineers apply various measures and tools for assessing software quality and indicating software weaknesses. Identifying such weaknesses helps software developers to improve the software quality in early stages instead of facing related problems after the software release.

Researchers have identified several software quality attributes and classified them into two categories: internal and external [1]. Internal quality attributes, such as cohesion, coupling, and complexity, are those that can be measured using design or code artifacts, which are typically available before the software is released. External quality attributes, such as maintainability, reliability, and testability, are those that depend on software environment factors that are determined during software use and cannot be measured using design and software artifacts. Therefore, these attributes cannot be measured before the software is released, although they are the ones of interest for software users. To overcome this problem, researchers have studied the relationship between internal and external quality attributes and have construct statistical-based models that use internal quality measures to estimate or predict external quality attributes. In this case, software engineers can apply these constructed models before releasing software to predict external quality attributes and to take necessary actions to improve the software quality.

Several empirical studies investigated the ability of design and code-based measures to estimate class fault-proneness (e.g., [2]-[10]). Some of these studies obtained different, and sometimes conflicting, results due to the different characteristics of the selected systems. Generalizing the results requires performing several replication studies. In this paper, we report a replication study exploring the impact of six design quality measures known as Chidamber and Kemerer (CK) measures [11] on class fault-proneness using five Java open-source systems of various domains and sizes. In addition, we built statistical-based prediction models based on the measures considered individually and in combination.

This paper is organized as follows. Section 2 provides an overview of the CK measures and a summary of some existing results. Section 3 describes the considered systems, the data collection process, and the analysis method. Sections 4 and 5 report and discuss the empirical study results. Finally, Section 6 concludes the paper and discusses future work.

2. Related Work

Researchers have proposed several measures that quantify various quality attributes. CK measures are the most frequently studied and referenced suite of measures [12]. This suite of measures include the weighted methods per class (WMC), depth of inheritance tree (DIT), response for a class (RFC), number of children (NOC), coupling between object classes (CBO), and a lack of cohesion between methods (LCOM). Several studies investigated these measures' abilities to predict different external quality attributes, such as fault-proneness (e.g., [2]-[10]), reuse-proneness (e.g., [1], [13]), and maintainability (e.g., [14], [15]). In addition, several studies explored the abilities of the statistical models that use these measures or some of them to predict refactoring opportunities (e.g., [16]-[19]).

Table 1 summarizes the direction of the impact of the CK measures on fault-proneness as reported in 17 existing studies, where "0" indicates that the measure was found to be a statistically insignificant class fault-proneness predictor. A positive (resp., negative) sign indicates that the models based on the measure were found to be statistically significant fault-proneness predictors, and the relationship between the measure and fault-proneness was found to be positive (resp., negative).

Table 1. Summary of Empirical Studies of CK Measures

Empirical study	WMC	DIT	NOC	CBO	RFC	COM
[8]	+/0	+	-	+	+	0
[26]	+	+	-	+	+	0
[27]	+	0		+	+	
[28]	+	-	0	+	+	+
[9]		0	0			
[29]	+/0	-/0		0		
[30]	+	+	0	+	+	+
[31]	+	+/0	+/0	+/0	+	+/0
[32]	+	+	+	+	+	+
[33]	+			+	+	+
[34]	+	+/0	0	+	+	
[2]	+	0	0	+	+	+
[35]	+	0	0	+	+	
[36]	0			0		
[10]	+					
[37]	+	0	-/0	+	+	+/0
[7]	+	-	+	+	+	+

3. Design of the Empirical Study

The goal of the empirical study was to explore the ability of the CK measures considered both individually and in combination to predict the fault-proneness of object-oriented classes. To perform the

empirical study, we randomly chose five Java open-source systems of various application domains. The first system is Art of Illusion v.2.5.1 [20] released on October 22, 2007. It is a three-dimensional (3D) rendering and modeling studio application consisting of 471 classes. The second system is DrJava v.beta-20090505-r4932 [21], which was released on May 5, 2009. It is a Java development environment consisting of 1036 classes. The third system is Eclipse v.1.0.1 [22], which was released on September 8, 2009. It is a multi-language software development environment consisting of 1438 classes. The fourth system is FreeMind v.0.8.1 [23], which was released on February 26, 2008. It is a mind-mapping software including 223 classes. The fifth system is JHotDraw v.7.4.1 [24], which was issued on January 17, 2010. It is a graphical user interface framework with 576 classes.

Building statistical prediction models requires collecting two pieces of data for each class in each of the selected systems. The first piece of data is CK values, and the second piece of data is the data related to whether the class has a fault detected during the maintenance phase. To collect the first piece of data, we used the CKJM tool [25]. To collect the second piece of data, we manually traced the maintenance history of the classes reported in the Concurrent Versions System (CVS) from Sourceforge.

To build the prediction models, we applied logistic regression analysis (LRA) [38], which is a statistical technique based on maximum likelihood estimation. The considered variables are classified into dependent and independent. In LRA, one or more independent variables are used to predict the dependent variable. The dependent variable has a binary value. LRA is called univariate LRA when one independent variable is considered in the analysis, and it is called multivariate LRA when two or more independent variables are considered.

In our context, CK measures were the independent variables, and the variable related to whether the class was found faulty was the dependent variable; it had a value of 1 if the class had at least one fault, and a value of 0 otherwise. The probability that a class is estimated as faulty is obtained using the following formula:

$$\pi(X_1, X_2, \ldots, X_n) = \frac{1}{1+e^{-(C_0+C_1X_1+C_2X_2+\ldots+C_nX_n)}} \quad (1)$$

where Xis are the CK measures, and the Ci coefficients are estimated using LRA. In univariate models, the absolute value of C1 indicates the strength of impact (positive or negative, according to the sign of C1) of the independent variable on the probability of the class to be faulty. In addition, the sign of C1 indicates whether the independent variable has a positive or negative impact on the probability of the class to be faulty. The p-value is the probability that the coefficient is different from zero by chance. In our analysis, we considered a typical significance threshold (α=0.05) to determine whether a measure was a statistically significant fault predictor.

To test the classification performance of a prediction model, we applied the above formula to obtain the value for each class. A threshold was considered to classify the class as estimated to be faulty if the was greater than or equal to the threshold value, or not faulty otherwise. With knowledge of the actual "faulty" values of the classes, the prediction models were constructed, and the classes in the system were accordingly classified based on the confusion matrix provided in Table 2.

Table 2. Confusion Matrix

		Actual faulty		
		0	1	
Estimated faulty	0	True Negatives (*tn*)	False Negatives (*fn*)	Estimated Negatives
	1	False Positives (*fp*)	True Positives (*tp*)	Estimated Positives
		Actual Negatives	Actual Positives	

In our analysis, we considered the following classification performance criteria:

- Recall = tp/(fn + tp);
- Precision = tp/(tn + fn);
- Fmeasure = 2×Precision×Recall/(Precision + Recall);

In addition, for each constructed prediction model, we obtained the area under the receiver operating characteristic curve (AUC) [38], which assesses the classification performance of the prediction model regardless of any threshold.

4. Univariate Analysis Results

Using the data of the classes in each system, we built a univariate prediction model based on each of the six CK measures. Therefore, the total number of constructed models is 30 (i.e., 5 systems × 6 measures). The coefficients and classification performance results of each of the constructed models are reported in Tables 3-7.

Table 3. Univariate Logistic Regression Results for Classes of Art of Illusion System

Measure	C_0	C_1	p-value	Recall	Precision	Fmeasure	AUC
WMC	-2.909	0.068	< 0.01	0.609	0.333	0.431	0.778
DIT	-2.656	0.354	< 0.01	0.493	0.337	0.400	0.698
NOC	-1.760	-0.006	0.905	0.942	0.146	0.253	0.515
CBO	-2.470	0.025	< 0.01	0.623	0.350	0.448	0.763
RFC	-2.932	0.018	< 0.01	0.667	0.348	0.458	0.804
LCOM	-2.065	0.003	< 0.01	0.449	0.403	0.425	0.727

Table 4. Univariate Logistic Regression Results for Classes of Drjava System

Measure	C_0	C_1	p-value	Recall	Precision	Fmeasure	AUC
WMC	-1.706	0.014	< 0.01	0.471	0.417	0.442	0.751
DIT	-1.649	0.063	0.175	0.361	0.160	0.222	0.493
NOC	-1.487	0.000	0.997	1.000	0.184	0.311	0.503
CBO	-1.963	0.032	< 0.01	0.508	0.458	0.481	0.756
RFC	-2.154	0.018	< 0.01	0.602	0.509	0.552	0.823
LCOM	-1.497	0.000	0.153	0.147	0.538	0.230	0.708

Table 5. Univariate Logistic Regression Results for Classes of Eclipse System

Measure	C_0	C_1	p-value	Recall	Precision	Fmeasure	AUC
WMC	-1.041	0.055	< 0.01	0.519	0.689	0.592	0.750
DIT	-0.887	0.290	< 0.01	0.367	0.534	0.435	0.593
NOC	-0.366	0.173	< 0.01	0.255	0.633	0.364	0.577
CBO	-1.009	0.056	< 0.01	0.517	0.708	0.598	0.751
RFC	-1.125	0.024	< 0.01	0.543	0.717	0.618	0.774
LCOM	-0.451	0.001	< 0.01	0.317	0.747	0.445	0.709

Table 6. Univariate Logistic Regression Results for Classes of Freemind System

Measure	C_0	C_1	p-value	Recall	Precision	Fmeasure	AUC
WMC	-2.309	0.044	< 0.01	0.400	0.326	0.359	0.695
DIT	-1.817	0.053	0.636	0.457	0.176	0.254	0.557
NOC	-1.685	0.010	0.922	0.171	0.240	0.200	0.541
CBO	-2.169	0.030	< 0.01	0.343	0.245	0.286	0.609
RFC	-2.679	0.021	< 0.01	0.486	0.327	0.391	0.721
LCOM	-1.924	0.001	< 0.01	0.286	0.476	0.357	0.719

Table 7. Univariate Logistic Regression Results for Classes of Jhotdraw System

Measure	C_0	C_1	p-value	Recall	Precision	Fmeasure	AUC
WMC	-0.080	0.054	< 0.01	0.457	0.713	0.557	0.627
DIT	0.056	0.173	< 0.01	0.518	0.732	0.607	0.557
NOC	0.431	0.225	0.017	0.164	0.766	0.271	0.496
CBO	-0.249	0.079	< 0.01	0.521	0.779	0.624	0.703
RFC	-0.212	0.023	< 0.01	0.490	0.762	0.597	0.660

LCOM	0.291	0.003	< 0.01	0.304	0.807	0.441	0.610

The results provided in Tables 3–7 led to the following observations:

1) Most of the models were found to be statistically significant predictors of fault-proneness. Specifically, the WMC, CBO, and RFC were always found to be statistically significant predictors of fault-proneness, and the DIT and LCOM were mostly found to be statistically significant predictors of fault-proneness. In contrast, the NOC was found to be a statistically insignificant predictor of fault-proneness most of the time. The next observations consider only the statistically significant predictor models.
2) All of the CK measures had a positive impact on fault-proneness. That is, increasing the complexity, coupling, depth of inheritance relations, and lack of cohesion caused the class to be more prone to faults.
3) The inheritance measures (i.e., the DIT and NOC) were always found to have the strongest impact on class fault-proneness, whereas the lack of cohesion measure (i.e., the LCOM) was always found to have the weakest impact on class fault-proneness. The complexity (i.e., the WMC) and coupling (i.e., the CBO and REF) measures alternated between the second and third positions when ordering the measures from the strongest to weakest impact on class fault-proneness.
4) Based on the Fmeasure and AUC values, the univariate prediction models based on REF were found to be the best classifier for classifying the classes into faulty and non-faulty most of the time. They had Fmeasure and AUC values that ranged within the interval of [0.391, 0.618] and [0.66, 0.823] respectively. The univariate prediction models based on inheritance measures were found to be the worst classifiers most of the time. The Fmeasure and AUC values for statistically significant models were based on the DIT range within the interval of [0.4, 0.607] and [0.557, 0.698] respectively. The Fmeasure and AUC values for statistically significant models were based on the NOC range within the interval of [0.271, 0.364] and [0.496, 0.577] respectively. Based on the average values of Fmeasure, the measures were ordered from the best to the worst corresponding model classifier as follows: the REF, CBO, DIT, WMC. LCOM, and NOC. In addition, based on the average values of the AUC, the measures were ordered from the best to the worst corresponding model classifier as follows: the REF, WMC, CBO, LCOM, DIT, and NOC.

5. Multivariate Analysis Results

Table 8. Multivariate Logistic Regression Results

System	Art of Illusion	DrJava	Eclipse	FreeMind	JHotDraw
c_0	-3.739	-2.325	-1.866	-2.679	-0.442
WMC	0.053	-0.065	0.025	-	-
DIT	0.304	-	0.263	-	-
NOC	-	-	-	-	-
CBO	0.011	-	0.019	-	0.052
RFC	-	0.049	0.015	0.021	0.014
LCOM	-	-	< 0.001	-	-
Recall	0.710	0.644	0.598	0.486	0.540
Precision	0.350	0.554	0.706	0.327	0.798
Fmeasure	0.469	0.596	0.648	0.391	0.645
AUC	0.817	0.799	0.784	0.721	0.716

For each of the five considered systems, we constructed a multivariate prediction model based on the values of the CK measures and fault data. When building a model, we proceeded in a stepwise fashion via

the backward elimination of independent variables. We started with the entire set of independent variables, and we removed the independent variable whose coefficient was not statistically significant and was associated with the largest p-value. In the following step, we built a model based on the set of remaining independent variables, and we removed the variable that was not statistically significant and had the largest p-value. We continued this stepwise removal of independent variables until the remaining independent variables were all statistically significant, and we used them to build the final model. The coefficients and classification performance results of the constructed models are provided in Table 8.

The results reported in Table 8 provide evidence that CK measures can be used in building multivariate prediction models with acceptable and sometimes excellent abilities to classify classes into faulty and non-faulty. Specifically, the results showed that the Fmeasure and AUC values for the constructed models ranged within the interval of [0.391, 0.648] and [0.716, 0.817] respectively. The averages of the Fmeasure and AUC values were 0.55 and 0.767.

6. Conclusion

In this paper, we reported the results of an empirical study that investigated the abilities of CK measures, considered individually and in combination, to predict class fault-proneness. Five open-source Java systems were considered in the study. The key results of the study provide evidence that in most cases, CK measures can be used in building models that are statistically significant predictors of class fault-proneness. All of the constructed multivariate models were found to be acceptable classifiers for the classes as faulty and non-faulty. The univariate models based on CK measures were found to have different prediction abilities, where models based on the RFC measure were mostly found to be the best predictors. The direction of the impact of the CK measures on fault-proneness was found to be positive most of the time, which matches the majority of the results reported in the literature.

All of the selected systems are open-source Java systems. To generalize the results, more studies considering systems developed using various programming languages and including industrial systems should be performed.

References

[1] Dallal, J. A., & Morasca, S. (2014). Predicting object-oriented class reuse-proneness using internal quality attributes. *Empirical Software Engineering*, 19(4), 775-821.

[2] Aggarwal, K., Singh, Y., Kaur, A., & Malhotra, R. (2009). Empirical analysis for investigating the effect of object-oriented metrics on fault proneness: A replicated case study. *Software Process Improvement and Practice*, *14*, 39-62.

[3] Dallal, J. A. (2015). Accounting for data encapsulation in the measurement of object-oriented class cohesion. *Journal of Software: Evolution and Process*, *27(5)*, 373-400.

[4] Dallal, J. A. Fault prediction and the discriminative powers of connectivity-based object-oriented class cohesion metrics. *Information and Software Technology*, *54(4)*, 396-416.

[5] Dallal, J. A. (2017). Predicting fault-proneness of reused object-oriented classes in software post-releases. *Arabian Journal of Science and Engineering*.

[6] Dallal, J. A., & Briand, L. (2010). An object-oriented high-level design-based class cohesion metric. *Information and Software Technology*, *52(12)*, 1346-1361.

[7] Dallal, J. A., & Morasca, S. (2018). Investigating the impact of fault data completeness over time on predicting class fault-proneness. *Information and Software Technology (Elsevier)*, *95*, 86-105.

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

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

[10] Zhou, Y., Xu, B., & Leung, H. (2010). On the ability of complexity metrics to predict fault-prone classes in object-oriented systems. *The Journal of Systems and Software*, *83*, 660-674.

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

[12] Saxena, P., & Saini, M. (2011). Empirical studies to predict fault proneness: a review. *International Journal of Computer Applications*, *22(8)*, 41-45.

[13] Dallal, J. A. (2014). The effects of incorporating special methods into cohesion measurement on class instantiation reuse-proneness prediction. *IET Software*, *8(6)*, 285-295.

[14] Dallal, J. A. (2013). Object-oriented class maintainability prediction using internal quality attributes. *Information and Software Technology*, *55(11)*, 2028-2048.

[15] Dagpinar, M., & Jahnke, J. H. (2003). Predicting maintainability with object-oriented metrics — An empirical comparison. *Proceedings of the 10th Working Conference on Reverse Engineering*.

[16] Dallal, J. A. (2012). Constructing models for predicting extract subclass refactoring opportunities using object-oriented quality metrics. *Information and Software Technology*, *54(10)*, 1125-1141.

[17] Dallal, J. A. (2017). Predicting move method refactoring opportunities in object-oriented code. *Information and Software Technology*, *92*, 105–120.

[18] Dallal, J. A., & Abdin, A. (2018). Empirical evaluation of the impact of object-oriented code refactoring on quality attributes: A systematic literature review. *IEEE Transactions on Software Engineering*, *44(1)*, 44-69.

[19] Kosker, Y., Turhan, B., & Bener, A. (2009). An expert system for determining candidate software classes for refactoring. *Expert Systems with Applications*, *36(6)*, 10000-10003.

[20] Art of Illusion. Retrieved June 2017 from http://sourceforge.net/projects/aoi/

[21] DrJava. Retrieved June 2017 from http://sourceforge.net/projects/drjava/

[22] Eclipse. Retrieved June 2017 from http://www.eclipse.org/

[23] FreeMind. Retrieved June 2017 from http://freemind.sourceforge.net/wiki/index.php/Main_ Page

[24] JHotDraw. Retrieved June 2017 from http://sourceforge.net/projects/jhotdraw/

[25] CKJM — Chidamber and Kemerer Java Metrics. Retrieved June 2017 from http://www.spinellis.gr/sw/ckjm/

[26] Briand, L. C., Wüst, J., Daly, J., & Porter, V. (2000). Exploring the relationship between design measures and software quality in object-oriented systems. *Journal of System and Software*, *51(3)*, 245-273.

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

[28] Briand, L. C., Wüst, J., & Lounis, H. (2001). Replicated case studies for investigating quality factors in object-oriented designs. *Empirical Software Engineering*, *6(1)*, 11-58.

[29] Subramanyam, R., & Krishnan, M. (2003). Empirical analysis of CK metrics for object-oriented design complexity: Implications for software defects. *IEEE Transactions on Software Engineering*, *29(4)*, 297-310.

[30] Gyimothy, T., Ferenc, R., & Siket, I. (2005). Empirical validation of object-oriented metrics on open source software for fault prediction. *IEEE Transactions on Software Engineering*, *3(10)*, 897-910.

[31] Olague, H., Etzkorn, L., Gholston, S., & Quattlebaum, S. (2007). 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*, *33(6)*, 402-419.

[32] English, M., Exton, C., Rigon, I., & Cleary, B. (2009). Fault detection and prediction in an open-source

software project. *Proceedings of the 5th International Conference on Predictor Models in Software Engineering*.

[33] Holschuh, T., Pauser, M., Herzig, K., Zimmermann, T., Premraj, R., & Zeller, A. (2009). Predicting defects in SAP Java code: an experience report. *Proceedings of the 31st International Conference on Software Engineering*, Companion Volume.

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

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

[36] Thapaliyal, M., & Verma, G. (2010). Software defects and object oriented metrics-an empirical analysis. *International Journal of Computer Applications*, *9(5)*, 41-44.

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

[38] Hosmer, D., & Lemeshow, S. (2000). *Applied Logistic Regression*, John Wiley and Sons.

Jehad Al Dallal is a professor and chairman of Information Science Department at Kuwait University. He received his PhD in computer science from the University of Alberta in Canada and was granted the award for best PhD researcher. Prof. Al Dallal has completed many research projects in the areas of software testing, software metrics, software refactoring, and communication protocols. In addition, he has published more than 100 papers in reputable journals and conference proceedings. Prof. Al Dallal was involved in developing more than 20 software systems. He also served as a technical committee member of several international conferences and as an associate editor for several refereed journals.