International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:10, No:6, 2016

Reasons for Non-Applicability of Software Entropy Metrics for Bug Prediction in Android

Arvinder Kaur, Deepti Chopra

***Abstract*—**Software Entropy Metrics for bug prediction have been validated on various software systems by different researchers. In our previous research, we have validated that Software Entropy Metrics calculated for Mozilla subsystem's predict the future bugs reasonably well. In this study, the Software Entropy metrics are calculated for a subsystem of Android and it is noticed that these metrics are not suitable for bug prediction. The results are compared with a subsystem of Mozilla and a comparison is made between the two software systems to determine the reasons why Software Entropy metrics are not applicable for Android.

***Keywords*—**Android, bug prediction, mining software repositories, Software Entropy.

I. Introduction

BUG prediction involves using the past characteristics of the software systems to determine the future bugs in the software system. Software Entropy Metrics proposed by Hassan [1] are used to quantify the complexity of source code changes. A change is made to the software for (i) bug correction, (ii) enhancement, and (iii) maintenance purposes. The most complex of these are the enhancement related changes which contribute to the complexity of code change process. This complexity is quantified in terms of Software Entropy given by (1):

$$S.E._n(P) = -\sum_{i=1}^{n}(P_i * log_2 P_i) \quad (1)$$

where, P_i is the probability of changes in the i^{th} file defined as the number of changes in i^{th} file divided by the total number of changes in all files of the software system/subsystem.

Hassan [1] validated the applicability of Software Entropy metrics using Simple Linear Regression (SLR) on six open source software systems including NetBSD, FreeBSD, OpenBSD, Postgre, KDE and KOffice.

Singh and Chaturvedi [4] employed the Software Entropy Metrics given by Hassan [1] for predicting bugs using Support Vector Regression (SVR). They validated the results for three subsystems of Mozilla and came to the conclusion that SVR performs better than SLR. Kaur and Kaur [10] have employed various statistical methods for prediction of software maintainability. Singh et al. [11] have compared models for predicting fault proneness in object oriented software systems.

Arvinder Kaur, Professor, is with the University School of Information and Communication Technology, Guru Gobind Singh Indraprastha University, New Delhi, Delhi, India (e-mail: arvinder70@gmail.com).

Deepti Chopra is with the University School of Information and Communication Technology, Guru Gobind Singh Indraprastha University, New Delhi, Delhi, India (e-mail: dchopra27@gmail.com).

We have also conducted previous studies [2], [3] to validate the applicability of Software Entropy Metrics for bug prediction in subsystems of Mozilla. In 'Entropy based Bug Prediction using Neural Network based Regression' [2], we verified that the Software Entropy Metrics are reasonably good predictors of bugs using a neural network based regression method to verify the results. Also, in 'Application of Locally Weighted Regression for predicting Faults using Software Entropy Metrics' [3], we show that Locally Weighted Regression (LWR) gives better bug prediction results compared to SVR.

In this study, we try to analyze why the Software Entropy Metrics are not able to predict the bugs in subsystems of Android. We try to compare subsystems of Mozilla and Android to understand the differences between the two software systems that may affect the applicability of Software Entropy Metrics.

This paper consists of the following sections: Section II describes how the Software Entropy metrics are calculated and also discusses regression results for the subsystems of Mozilla and Android. Section III compares the differences between Mozilla and Android to understand why Software Entropy metrics are not able to predict bugs in Android subsystem. In Section IV, the differences analyzed are concluded to generalize the findings.

II. Software Entropy Metrics

The Software Entropy for a particular period is calculated by using the formula given in (1). The Software Entropy $S.E._n$ is normalized by using (2), such that $0 \leq S.E. \leq 1$. The normalized Software Entropy makes it possible to compare the Software Entropy for subsystems containing different number of files.

$$S.E(P) = \frac{1}{Maximum\ Entropy} * S.E._n(P) = -\frac{1}{\log_2 n} * \sum_{k=1}^{n}(P_i * \log_2 P_i) \quad (2)$$

where $P_i \geq 0,\ \forall i \in 1, 2, ..., n$ and $\sum_{i=1}^{n} P_i = 1$

After calculating the normalized Software Entropy, each file is assigned a complexity value. In general, greater the complexity value, more buggy is the file. History Complexity Metric (HCM) is calculated for each file in the software system.

For a period k, with entropy $S.E._k$ where a set of files, F_k are modified with a probability P_j for each file $j \in F_k$, (3) defines the History Complexity Period Factor ($HCPF_k$) for a file j during period k.

$$HCPF_k(j) = C_{kj} * S.E._k, \quad j \in F_k \quad (3)$$

where, C_{kj} is the contribution of Software Entropy for period k ($S.E._k$) that is assigned to file j. By varying the value of C_{kj}, three variants of *HCPF* are obtained to calculate HCM. Fig. 1 defines the different variants of HCM.

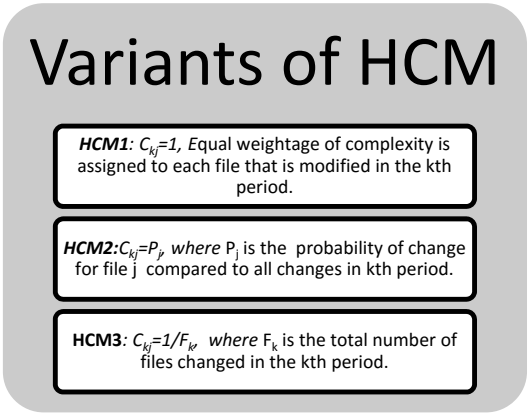

Fig. 1 HCM metrics

HCM value for a file j over the evolution period $\{x,.., y\}$ is calculated using (4):

$$HCM_{\{x,..,y\}}(j) = \sum_{k\in\{x,..,y\}} HCPF_k(j) \quad (4)$$

The value of HCM and so the complexity increases with time. HCM value for a software subsystem (S) over the evolution periods {m,…,n} is the sum of HCMs for each file in the subsystem as defined in (5).

$$HCM_{\{x,...,y\}}(S) = \sum_{j\in S} HCM_{\{x,...,y\}}(j) \quad (5)$$

HCM metrics, Normalized Software Entropy and number of changes and faults per year for the subsystems of Android and Mozilla are listed in Tables I and II, respectively.

TABLE I
ENTROPY FOR SUBSYSTEMS OF ANDROID

Subsystem	Android/platforms_frameworkbase /location			Android/platforms_frameworkbase/ keystore			Android/platforms_frameworkbase/ obex			Android/platforms_frameworkbase/ native		
Year	Changes	Faults	Normalized Entropy	Changes	Faults	Normalized Entropy	Changes	Faults	Normalized Entropy	Changes	Faults	Normalized Entropy
2009	13	6	0.412	6	2	0.539	36	28	0.962	-	-	-
2010	31	13	0.634	4	6	0.239	2	0	0.227	27	19	0.885
2011	5	4	0.304	16	11	0.624	2	3	0.227	7	5	0.575
2012	25	54	0.573	8	8	0.419	0	2	0	7	10	0.681
2013	21	10	0.584	5	25	0.461	3	6	0.361	2	6	0.27

TABLE II
ENTROPY FOR SUBSYSTEMS OF MOZILLA

Subsystem	Mozilla/layout/forms			Mozilla/layout/generic		
Year	Changes	Faults	Normalized Entropy	Changes	Faults	Normalized Entropy
2007	37	127	0.901	138	834	0.849
2008	2	131	0.184	73	993	0.633
2009	2	247	0.184	58	1315	0.558
2010	2	245	0.184	61	1173	0.555
2011	116	296	0.857	504	1465	0.85
2012	222	461	0.819	1005	2058	0.821
2013	3	660	0.169	22	2119	0.546

The subsystems "Mozilla/layout/forms/" and "Mozilla/layout/generic/" of Mozilla, and "Android/platform_frameworks_base/location/","Android/platform_frameworks_base/keystore/","Android/platform_frameworks_base/obex/" and "Android/platform_frameworks_base/native/" of Android are used to validate the Software Entropy metrics for bug prediction. The data are collected from Mozilla-central [5] and GitHub [6] for Mozilla and Android respectively.

Regression analysis is done to determine the performance of calculated metrics for bug prediction. The tool used to perform regression is Weka 3.6 [7], which is a popular tool for data mining and machine learning. The results are compared based on the values of Correlation Coefficient, Mean Absolute Error (M.A.E.) and Root Mean Square Error (R.M.S.E.). These measures as defined in [8] are:

Let $p_1, p_2, \ldots, p_n$ be the predicted values and $a_1, a_2, \ldots, a_n$ are the actual values.

- **Correlation Coefficient:** measures the statistical correlation between the a's and the p's given by (6):

$$Correlation\ Coefficient = \frac{s_{pa}}{s_p * s_a} \quad (6)$$

where $s_{pa} = \frac{\sum_i (p_i-\bar{p})(a_i-\bar{a})}{n-1}$, $s_p = \frac{\sum_i (p_i-\bar{p})^2}{n-1}$ and $s_{pa} = \frac{\sum_i (a_i-\bar{a})^2}{n-1}$.

- **Mean Absolute Error (M.A.E.):** as in (7) averages the magnitude of individual errors. The sign of error is not considered.

$$M.A.E. = \frac{|p_1 - a_1| + \cdots + |p_n - a_n|}{n} \quad (7)$$

- **Root Mean Square Error (R.M.S.E.):** measures the differences between the predicted and actual values of the samples as given in (8).

$$\sqrt{\frac{(p_1 - a_1)^2 + \cdots + (p_n - a_n)^2}{n}} \quad (8)$$

The regression results for these subsystems are listed in Tables III-VIII.

TABLE III
REGRESSION RESULTS FOR "MOZILLA/LAYOUT/FORMS/"

Model	Metrics	Correlation Coefficient	M.A.E	R.M.S.E
Support Vector Regression (SVR)	HCM1	0.677	104.144	125.795
	HCM2	0.708	92.497	119.777
	HCM3	0.708	92.496	119.777
Locally Weighted Regression (LWR)	HCM1	0.7233	109.149	127.975
	HCM2	0.7237	109.367	128.414
	HCM3	0.7238	109.367	128.415
Multilayer Perceptron	HCM1	0.701	100.15	128.801
	HCM2	0.738	92.114	123.628
	HCM3	0.738	92.115	123.627

TABLE IV
REGRESSION RESULTS FOR "MOZILLA/LAYOUT/GENERIC/"

Model	Metrics	Correlation Coefficient	M.A.E	R.M.S.E
Support Vector Regression (SVR)	HCM1	0.943	125.089	173.618
	HCM2	0.898	186.815	199.650
	HCM3	0.899	186.709	199.507
Locally Weighted Regression (LWR)	HCM1	0.921	135.910	177.004
	HCM2	0.925	136.631	171.991
	HCM3	0.925	136.623	171.974
Multilayer Perceptron	HCM1	0.950	118.841	140.277
	HCM2	0.925	154.611	174.334
	HCM3	0.925	154.636	174.367

TABLE V
REGRESSION RESULTS FOR
"ANDROID/PLATFORM_FRAMEWORKS_BASE/LOCATION/"

Model	Metrics	Correlation Coefficient	M.A.E	R.M.S.E
Support Vector Regression (SVR)	HCM1	0.014	17.711	22.076
	HCM2	-0.005	16.749	21.310
	HCM3	-0.005	16.750	21.310
Locally Weighted Regression (LWR)	HCM1	-0.663	26.745	31.2622
	HCM2	-0.591	25.181	29.429
	HCM3	-0.591	25.181	29.429
Multilayer Perceptron	HCM1	-0.832	19.841	25.022
	HCM2	-0.911	20.602	25.395
	HCM3	-0.910	20.601	25.395

It is seen that the correlation coefficients for three out of four subsystems of Android are negative, whereas those for subsystem of Mozilla are positive. This indicates that as the Software Entropy; i.e., the complexity of code changes increases the number of bugs in the subsystem decreases for Android system. But, as we know that the number of bugs increases with the code change complexity [1], we try to understand the differences between Mozilla and Android. The goal of this study is to analyze why the Software Entropy metrics are not useful for predicting bugs in Android.

TABLE VI
REGRESSION RESULTS FOR
"ANDROID/PLATFORM_FRAMEWORKS_BASE/KEYSTORE/"

Model	Metrics	Correlation Coefficient	M.A.E	R.M.S.E
Support Vector Regression (SVR)	HCM1	0.377	6.787	7.293
	HCM2	0.654	4.881	5.830
	HCM3	0.654	4.881	5.829
Locally Weighted Regression (LWR)	HCM1	0.052	8.855	10.809
	HCM2	0.049	8.884	10.935
	HCM3	0.049	8.884	10.935
Multilayer Perceptron	HCM1	0.401	10.389	11.701
	HCM2	0.408	9.931	11.067
	HCM3	0.408	9.931	11.066

TABLE VII
REGRESSION RESULTS FOR "ANDROID/PLATFORM_FRAMEWORKS_BASE/OBEX/

Model	Metrics	Correlation Coefficient	M.A.E	R.M.S.E
Support Vector Regression(SVR)	HCM1	-0.574	3.136	3.805
	HCM2	-0.639	3.042	3.554
	HCM3	-0.638	3.042	3.554
Locally Weighted Regression (LWR)	HCM1	-0.337	2.926	3.105
	HCM2	-0.459	3.234	3.268
	HCM3	-0.459	3.234	3.268
Multilayer Perceptron	HCM1	-0.715	2.494	3.137
	HCM2	-0.828	2.683	3.191
	HCM3	-0.828	2.683	3.191

TABLE VIII
REGRESSION RESULTS FOR
"ANDROID/PLATFORM_FRAMEWORKS_BASE/NATIVE/"

Model	Metrics	Correlation Coefficient	M.A.E	R.M.S.E
Support Vector Regression(SVR)	HCM1	-0.605	2.812	3.984
	HCM2	-0.619	2.524	3.443
	HCM3	-0.619	2.524	3.443
Locally Weighted Regression (LWR)	HCM1	-0.694	2.5	3.535
	HCM2	-0.695	2.5	3.536
	HCM3	-0.695	2.5	3.536
Multilayer Perceptron	HCM1	-0.879	3.303	3.74
	HCM2	-0.854	3.384	3.839
	HCM3	-0.853	3.384	3.838

TABLE IX
COMPARISON PARAMETERS

Parameter	Description
Changes per year	The number of feature introducing changes that take place in the software system/subsystem per year.
Bugs per year	The number of bugs that are repaired in the software system/subsystem per year.
Software Entropy	The value of normalized Software Entropy as given in (2)
Software type	The type of software system
Development Approach	How is the software developed?
Source code management system	The repository/tool used for managing the source code
Bug tracking	How are bugs reported and managed?

III. Comparison of Mozilla and Android

In this section, we compare the two software: Mozilla and Android in order to understand the differences that make Software Entropy metrics non-applicable to Android systems. The two subsystems are compared on the parameters listed in Table IX. The subsections of this section compare the two systems based on these parameters in detail and the descriptive statistics for some of these parameters are listed in Table X.

A. Changes per Year

The number of changes per year, particularly feature introducing changes, are compared for the subsystems of Mozilla and Android. The mean numbers of changes per year for the subsystems of Mozilla (54.86,265.86) are higher than those for the subsystems of Android (19,7.8,8.6,10.75). The mean number of changes is higher in Mozilla, and also the variance for number of changes per year is very high in Mozilla in comparison to Android (see Table X). Thus, a low variance in the number of changes per year may be a factor contributing to the non-applicability of Software Entropy metrics for subsystems of Android.

TABLE X
Descriptive Statistics of the Comparison Parameters

Mozilla/layout/forms/						
Parameter	Range	Minimum	Maximum	Mean	Std. Deviation	Variance
Number of Changes	220	2	222	54.86	84.72	7177.48
Number of bugs	533	127	660	309.57	191.15	36536.62
Software Entropy	.732	.169	.901	.471	.364	.132
Mozilla/layout/generic/						
Parameter	Range	Minimum	Maximum	Mean	Std. Deviation	Variance
Number of Changes	983	22	1005	265.86	365.43	133537.14
Number of bugs	1285	834	2119	1422.43	499.23	249231.29
Software Entropy	.304	.545	.850	.687	.146	.021
Android/platform_frameworks_base/location/						
Parameter	Range	Minimum	Maximum	Mean	Std. Deviation	Variance
Number of Changes	26	5	31	19.00	10.19	104
Number of Bugs	50	4	54	17.40	20.76	430.8
Software Entropy	.330	.304	.634	.501	.138	.019
Android/platform_frameworks_base/keystore/						
Parameter	Range	Minimum	Maximum	Mean	Std. Deviation	Variance
Number of Changes	12	4	16	7.8	4.82	23.20
Number of Bugs	23.000	2	25	10.40	8.79	77.30
Software Entropy	.384	.240	.624	.456	.144	.021
Android/platform_frameworks_base/obex/						
Parameter	Range	Minimum	Maximum	Mean	Std. Deviation	Variance
Number of Changes	36	0	36	8.6	15.35	235.80
Number of Bugs	28	0	28	7.8	11.49	132.20
Software Entropy	.962	0	.962	.355	.363	.132
Android/platform_frameworks_base/native/						
Parameter	Range	Minimum	Maximum	Mean	Std. Deviation	Variance
Number of Changes	25	2	27	10.75	11.09	122.92
Number of Bugs	14	5	19	10	6.38	40.67
Software Entropy	.615	.270	.885	.603	.256	.066

B. Bugs per Year

The number of bugs per year for the subsystems of Mozilla and Android are compared. The number of bugs per year in Mozilla is large (in hundreds or thousands), while those for Android subsystems are less (see Table X). Thus, a low number of bugs repaired per year may be another factor contributing to the non-applicability of Software Entropy metrics for the subsystem of Android.

C. Software Entropy

The value of Normalized Software Entropy is for the subsystems of Android and Mozilla are plotted in Fig. 2 and Fig. 3. The Software Entropy for Mozilla subsystem is nearly constant for some time before it increases or decreases, while for Android subsystem the Software Entropy is rapidly changing. This rapidly changing Software Entropy may be a factor that renders Software Entropy inapplicable for predicting bugs in Android.

D. Software Type

Mozilla is a web browser, while Android is an Operating system (O.S.). A web browser is an application software used for accessing, fetching and presenting the information available on the internet. On the other hand, an O.S. is a system software that is responsible for managing and coordinating hardware and software resources. Android is a

system software that manages other application software, hence it is a more complex software system than Mozilla. The types of bugs in an O.S. are also a bit different from those in other software systems. An O.S. has more concurrency bugs than memory-based or semantic bugs as compared to other software systems [9]. This difference between the two software systems can be another factor that leads to non-applicability of Software Entropy Metrics for Android and not for Mozilla.

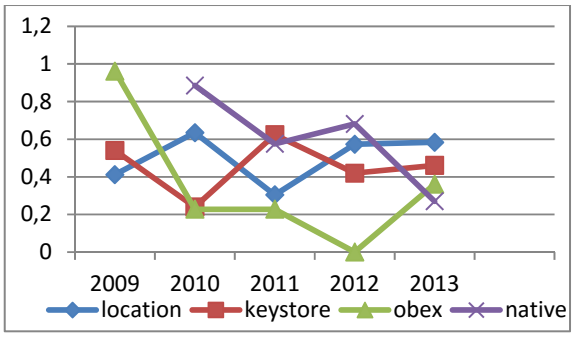

Fig. 2 Software Entropy for subsystems of Android

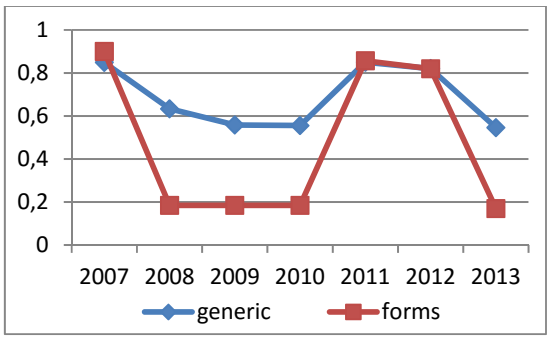

Fig. 3 Software Entropy for subsystems of Mozilla

E. Development Approach

Mozilla is developed by Open Source Community initiated by Netscape members in 1998. First major version was released in 2002, and the Mozilla Foundation supported by open source developers and various companies was created in 2003. It follows a community based development approach.

On the other hand, Android was initially developed by Android Inc. and later taken over by Google in 2005. It was unveiled in 2007 along with the formation of Open Handset Alliance. The Android source code is released under the open source license by Google, but the fact remains that Android is developed privately by Google and after the latest changes and updates are made, the source code is released publically.

It can be said that since Android is developed privately by one of the best companies in the world i.e. Google, the complexity of code changes is not the same as for other open source software systems. Hence, Software Entropy metrics are not applicable for Android.

F. Source Code Management System

Mozilla uses a Mercurial repository for managing its source code, keeping track of changes and sharing changes with others. Mozilla-central is a Mercurial repository that keeps track to changes made in the main development tree, whereas Android uses Git as its open source version control system. Repo is a repository management tool that is built over Git to handle multiple Git repositories at once. Thus, Mozilla has its own Mercurial repository for managing the source code, whereas Android uses the services of Git which is an open source software version control system. This is another difference that might have a slight impact on the non-applicability of Software Entropy metrics for bug prediction in Android.

G. Bug Tracking

Mozilla uses Bugzilla for maintaining its bug database and keeping track of the reported bugs. This is a web-based tool that was initially developed and used by the Mozilla Foundation. Bugzilla was later released as an open source software and many organizations have adopted it for using as a bug tracking system for open source as well as proprietary software. On the other hand, Android Open Source Project uses a public issue tracker for reporting bugs and requesting updates.

In Bugzilla, the sole purpose is to report and track bugs while providing complete details for reproducing a bug; however, with the Android issue tracker, you can also make request for features and updates. Hence, the fact that Android does not use different systems for handling bug reports and feature requests might affect the applicability of Software Entropy Metrics for bug prediction in Android.

IV. Conclusion and Future Work

In this study, we calculated Software Entropy Metrics for two subsystems of Mozilla and four subsystems of Android. We noticed negative correlation coefficients in all of the regression techniques for three out of four subsystems of Android, indicating that the complexity of code changes is not able to predict bugs in Android subsystem effectively. Therefore, in order to analyze the reason why Software Entropy metrics cannot be used for predicting bugs in Android, we compared the two software systems: Mozilla and Android.

On the basis of the comparison, we concluded that the factors contributing to non-applicability of Software Entropy Metrics for Android include: (i) low variance in the number of changes per year, (ii) low number of bugs repaired per year, (iii) rapidly changing Software Entropy, (iv) types and complexity of bugs in an O.S. are different from the ones in other application software, (v) private development approach rather than a community based development approach, (vi) use of an open source code management repository service and (vii) same system for handling both bug reports and feature requests.

We plan to enhance the study, by identifying the differences in types of changes/bugs in the software systems that may affect the applicability of Software Entropy metrics. Also, there is a need to replicate this study for other subsystems of Android.

REFERENCES

[1] A.E. Hassan, 2009. Predicting Faults based on complexity of code change. In the proceedings of 31st Intl. Conf. on Software Engineering. 2009. pp. 78-88.

[2] Arvinder Kaur, Kamaldeep Kaur, and Deepti Chopra."Entropy based Bug Prediction using Neural Network based regression," in *Computing, Communication & Automation (ICCCA), 2015 International Conference on*, vol., no., pp.168-174, 15-16 May 2015.

[3] Arvinder Kaur, Kamaldeep Kaur, and Deepti Chopra."Application of Locally Weighted Regression for Predicting Faults Using Software Entropy Metrics." In the *Proceedings of the Second International Conference on Computer and Communication Technologies*, pp. 257-266. Springer India, 2016.

[4] V.B. Singh and K.K.Chaturvedi. 2012. Entropy based Bug Prediction using Support Vector Regression. In the proceedings of 2012 12th International Conference on Intelligent Systems Design and Applications (ISDA). 2012. pp. 746-751.

[5] http://hg.mozilla.org/mozilla-central/file/9ee9e193fc48/layout/forms.

[6] https://github.com/android/platform_frameworks_base/tree/master/location.

[7] Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I. H.: The WEKA data mining software: an update. ACM SIGKDD explorations newsletter, 11(1), 10-18. (2009).

[8] Witten, Ian H., Eibe Frank. Data Mining: Practical machine learning tools and techniques. Morgan Kaufmann, 2005.

[9] Lin Tan ,Chen Liu, Zhenmin Li, Xuanhui Wang, Yuanyuan Zhou, Chengxiang Zhai. Bug Characteristics in Open Source Software.

[10] Kaur, Arvinder, and Kamaldeep Kaur. "Statistical comparison of modelling methods for software maintainability prediction." *International Journal of Software Engineering and Knowledge Engineering* 23.06 (2013): 743-774.

[11] Singh, Yogesh, Arvinder Kaur, and Ruchika Malhotra. "A comparative study of models for predicting fault proneness in object-oriented systems."*International Journal of Computer Applications in Technology* 49.1 (2014):22-41.