DOI: 10.24996/ijs.2021.SI.2.7

ISSN: 0067-2904

Software Fault Estimation Tool Based on Object-Oriented Metrics

Atica M. Altaie, Asmaa Yaseen Hamo*, Rasha Gh. Alsarraj
College of Computer Science and Mathematics, University of Mosul, Mosul, Iraq

Abstract
A fault is an error that has effects on system behaviour. A software metric is a value that represents the degree to which software processes work properly and where faults are more probable to occur. In this research, we study the effects of removing redundancy and log transformation based on threshold values for identifying faults-prone classes of software. The study also contains a comparison of the metric values of an original dataset with those after removing redundancy and log transformation. E-learning and system dataset were taken as case studies. The fault ratio ranged from 1%-31% and 0%-10% for the original dataset and 1%-10% and 0%-4% after removing redundancy and log transformation, respectively. These results impacted directly the number of classes detected, which ranged between 1-20 and 1-7 for the original dataset and 1-7 and 0-3) after removing redundancy and log transformation. The Skewness of the dataset was deceased after applying the proposed model. The classified faulty classes need more attention in the next versions in order to reduce the ratio of faults or to do refactoring to increase the quality and performance of the current version of the software.

Keywords *:* software, Fault predication, Threshold value, Remove redundancy, Log transformation.

أداة تخمين الخطأ للبرمجيات بالاعتماد على مقاييس البرمجة الموجهة

عاتكة محمد الطائي, اسماء ياسين حمو*, رشا غانم السراج
كلية علوم الحاسوب والرياضيات, جامعة الموصل, الموصل, العراق

الخلاصه
الخطأ هو الذي يؤثر على سلوك النظام. مقياس البرمجيات هو قيمة تمثل درجة عمليات البرمجيات التي تعمل بشكل صحيح و تصف اين الاخطاء المحتمل حدوثها. في هذا البحث، تم دراسة تأثير إزالة التكرار والتحويل اللوغاريتمي بناءً على قيم حد العتبة لتحديد فئات البرمجيات المعرضة للأخطاء ، كما يحتوي على مقارنة بين القيم لمجموعة البيانات الأصلية والقيم بعد إزالة التكرار والتحويل اللوغاريتمي. تم أخذ مجموعة بيانات التعلم الإلكتروني والنظام كحالات دراسية. وتراوحت نسبة الخطأ بين (1٪ -31٪) ، (0٪ -10٪) لمجموعة البيانات الأصلية و (1٪ -10٪) ، (0٪ -4٪) بعد إزالة التكرار والتحويل اللوغاريتمي وتأثير هذه النتائج بشكل مباشر على عدد الاصناف المكتشفة التي تراوحت بين (1-20) و (1-7) لمجموعة البيانات الأصلية و (1-7) و (0-3) بعد إزالة التكرار والتحويل اللوغاريتمي. انحراف مجموعة البيانات قل بعد تطبيق النموذج المقترح. الاصناف المصنفة بانها معرضة للخطأ تحتاج المزيد من الاهتمام في الإصدارات اللاحقة لتقليل نسبة الأخطاء أو القيام بإعادة الهيكلة لزيادة جودة وأداء الإصدار الحالي من البرمجيات.

*Email: asmahammo@uomosul.edu.iq

I. Introduction

Software systems have become very common, and because they depend on the programming of many people, there is a possibility of the emergence of errors. These errors may affect code quality [1], reliability [2] and maintainability [3]. After diagnosing the error in the classes, refactoring can be performed to increase accuracy of the applications [4].
Object-oriented metrics are used to measure software quality during software development and beyond. There are metrics that aid to measure the quality of the code and give suggestions to improve reuse, ease of maintenance, and detection errors, the most important of which are CK (Chidamber and Kemerer) metrics [5]. The CK metrics cover the most significant object-oriented properties, including cohesion, coupling, size, inheritance, and complexity [6]. They take high importance in applications that need accuracy. The used CK metrics are summarized in Table- 1 [7, 8].

Table 1- Metrics used in this study

Metric	Definition
WMC ("Weighted Methods per Class")	A measure representing the total complexity of class's methods.
DIT ("Depth of Inheritance Tree")	A measure representing the depth of a class diagram.
NOC ("Number Of Children")	A measure representing the number of classes associated with a class with an inheritance relationship.
CBO ("Coupling Between Object classes")	A measure representing the number of classes associated with a class with any relationship.
RFC ("Response For Class")	A measure representing the total number of methods of class taking into account received messages.
LCOM ("Lack Of Cohesion Metric")	A measure representing the correlation between methods and local variables of a certain class.

There are many previous research attempts to locate the error in the source code, which relied on different principles. Till now, no study and model have been identified that can be applied to all software and gives accurate results for all projects. Thus, this paper tried to have a different equation for each software to meet with it by adding a constant value, ranging from zero to one and then to the threshold value, as determined by the experiment to increase the accuracy of results.
In order to improve the quality of the program, these errors must be identified more precisely. In this study, an indicator is needed to reduce the effort to understand, maintain, and refactor. Therefore, the software metrics were used to determine which classes are needed more to focus on the testing and refactoring. After detecting the faulty classes, the refactoring process can be performed.
The present study aims to apply a methodology that focuses on removing redundancy and then applying log transformation to improve the quality of software metrics for identifying faults-prone classes of open source software and to view a comparison of the metric values of the original dataset with the values of the metric after performing the remove redundancy, log transformation, and recording of results.
The research is organized as follows. Section two describes related work. Section three describes methodology and contains a representation of UML (Unified Modelling Language) diagrams that shows the overall structure of the research. Section four introduces case studies, followed by the results that are obtained from the proposed methodology implementation. Finally, Sections five and six describe the conclusions and recommendations for the future of the research.

II. Related Work

In 2010, Shaik *et al.* demonstrated a study on the software metrics and their growing importance in software development and obtaining certain characteristics of the software [9]. In 2012, Ferreira *et a.l* studied open source projects, defined a threshold value for software metrics of object-oriented programs, and explored the importance of deriving a threshold value for the metrics of Software Engineering, which they used to enhance the quality of open source programs [10]. Also in the same year, Chawla utilized five software metrics for analysing a set of three sorting programs based on java.

There were 3 software measurement tools which were utilized for judging their performance in terms of metrics indicated. The tool's comparative analysis was specified for indicating how they are different in delivering results with regard to the same programs [11].

In 2013, Kapila *et al.* demonstrated all the approaches of faults prediction for getting aid with such work to design Bayesian inference and Logistic regression model. Also, it was indicated that the Bayesian inference graph might be provided for the probabilistic approach for faults identified and presented for the next upcoming release. With regard to the Probabilistic reliability analysis, the Bayesian inference was suggested for evaluating risk-related data. Such finding suggested a relation between object-oriented metrics and faulty classes [12].

In 2015, Shatnawi proposed a model which takes into consideration data distribution and skewness of the data and suggests log transformation and threshold value to identify faulty classes of open source programs. He showed the importance of using log transformation in fault predicting and its impact on the validity of the results [13].

In 2017, Zhang *et al.* studied the effects of the log, Box-Cox, and rank transformations on the normality of software metrics for open source software. Compared with their results, they selected the best training for a target open source software, then measured the performance based on it [14]. Gupta and Saxena proposed a model that used fourteen metrics from open source dataset and explored the importance of object oriented metrics and their relationship with bug prediction. The bug prediction was calculated based on equations that used these fourteen metrics [15]. Also in the same year, Gupta *et a.l* proposed a model to formulate some assumptions matching to each other and to object-oriented metrics , where the best appropriate metrics were chosen for the proposed model. The Logistic-Regression-Classifier offers precision among all classifiers. The proposed model was trained and tested on each of the dataset and the precision of the software was calculated in each case [16].

In 2018, three researches were done in this area, as follows.

Zhu and Pham identified the significant difference between software metrics and observed defect prediction. They also studied the relations involved in the object-oriented metrics "CK metrics suite" and the number of defects. They finally decided on the differences of the metrics, to eclipse classes as defective, and selected them with regard to defect prediction. They took a sample dataset from the bug prediction dataset of source code metrics as the data is based on Eclipse classes[17].

- Zhu *et a.l* suggested a theoretic software reliability model which incorporates the process of fault detection which is a stochastic process because of randomness resulting from environmental factors. The environmental factor, Percentage of Reused Modules, has been specified as gamma distribution in their work on the basis of collected data from the industry. Furthermore, the Open Source Software project data have been involved for showing the efficiency and predictive power regarding the suggested model[18].
- Belachew *et al.* assessed and analyzed software metrics utilized for measuring software quality, especially software products and processes. The software quality is used to measure how the software is developed and the way that software is in accordance with the design. Yet, the quality standard utilized from one organization was considered to be distinctive from the others. Thus, it was better to utilize the software metrics for measuring its quality as well as the quality of the majority of the current software metrics tools [19].

In 2019, Rahmann *et al*. applied the software change metrics with regard to defect prediction. Furthermore, the performances of excellent machine learning and hybrid algorithms were used in the estimation of the defect with changing metrics. Hybrid algorithms displayed improvements in performance, precision, and recall. The acquired results specified that GFS-logitboost-c has optimum defect prediction capability [20].

In the proposed model herein, the advantages of most previous studies were taken and combined in one model, starting with the metric collection, log transformation, and threshold value, while ending with skews calculation in each step and the use of these functions to identify faulty classes of open source projects.

III. Methodology

Figure-1 demonstrates the Use-Case diagram of the proposed tool to clarify the relationship of the system with the software engineer and the basic operation, as well as how to use it in detecting faulty classes in the software.

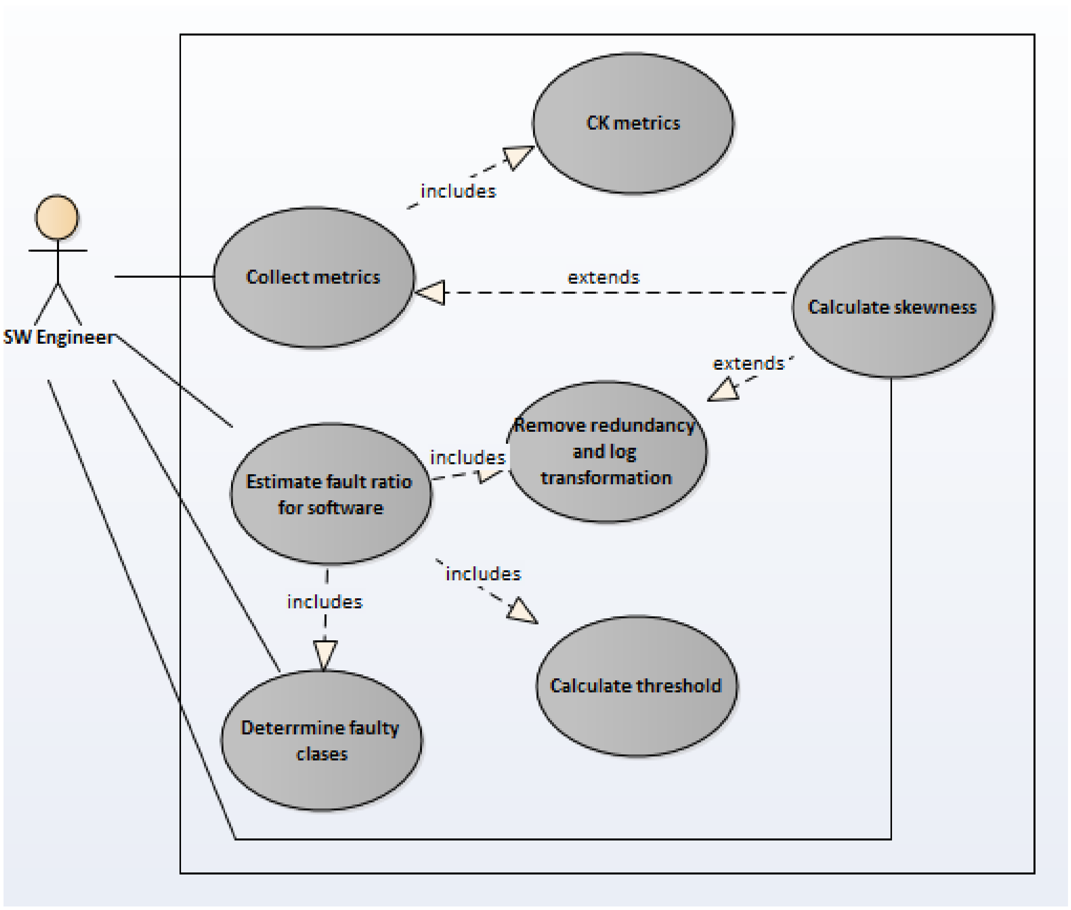

Figure 1- Use-Case diagram for the proposed model

The tool consists of seven classes, as follows.

- MainClass: It calls all classes of the proposed model, sends the variables, returns the results, and prints the final results at the program interface.
- MetricCollection: It opens the original file that contains data of many metrics collected, and extracts only six values, that represent the Chidamber and Kemerer's metrics suite used in the proposed model, in another file.
- RemoveRedundancy: It opens a file that contains CK metrics and removes redundancy from all records in the file.
- LogTransformation: It opens a file that contains CK metrics after removing redundancy, applies a log transformation for all values, and saves the results in another file.
- ThresholdCalculation: It takes the values of each metric in the file and computes the threshold through equation 1.

Threshold = mean (M) + standard deviation (SD) + constant value ….. Equation 1

It was supposed to add a constant value that ranged from 0 to 1 to the threshold equation. The higher value implies a higher accuracy of the results for the faulty classes. Values of 0.5 and 0.4 were selected, which were suitable for the selected software, while it was possible to increase or decrease the value for other types of software depending on the accuracy of the results required. Through experience, it was found that these values give the best results, and accordingly, they were relied upon. When the constant value is close to zero, the number of classes increases, while when it is close to one, the number of classes decreases. Thus, we can be more specific about the classes that need more attention by increasing the constant value.

- ClassesFaultyPresent: It computes the ratio of the fault classes depending on the number of classes that have a value higher than the threshold value divided by the total number of classes.
- SkewnessCalculation: It computes the skewness of the values for each metric and returns the results.

Figure-2 shows an activity diagram of the proposed model to clarify the interaction between processes.

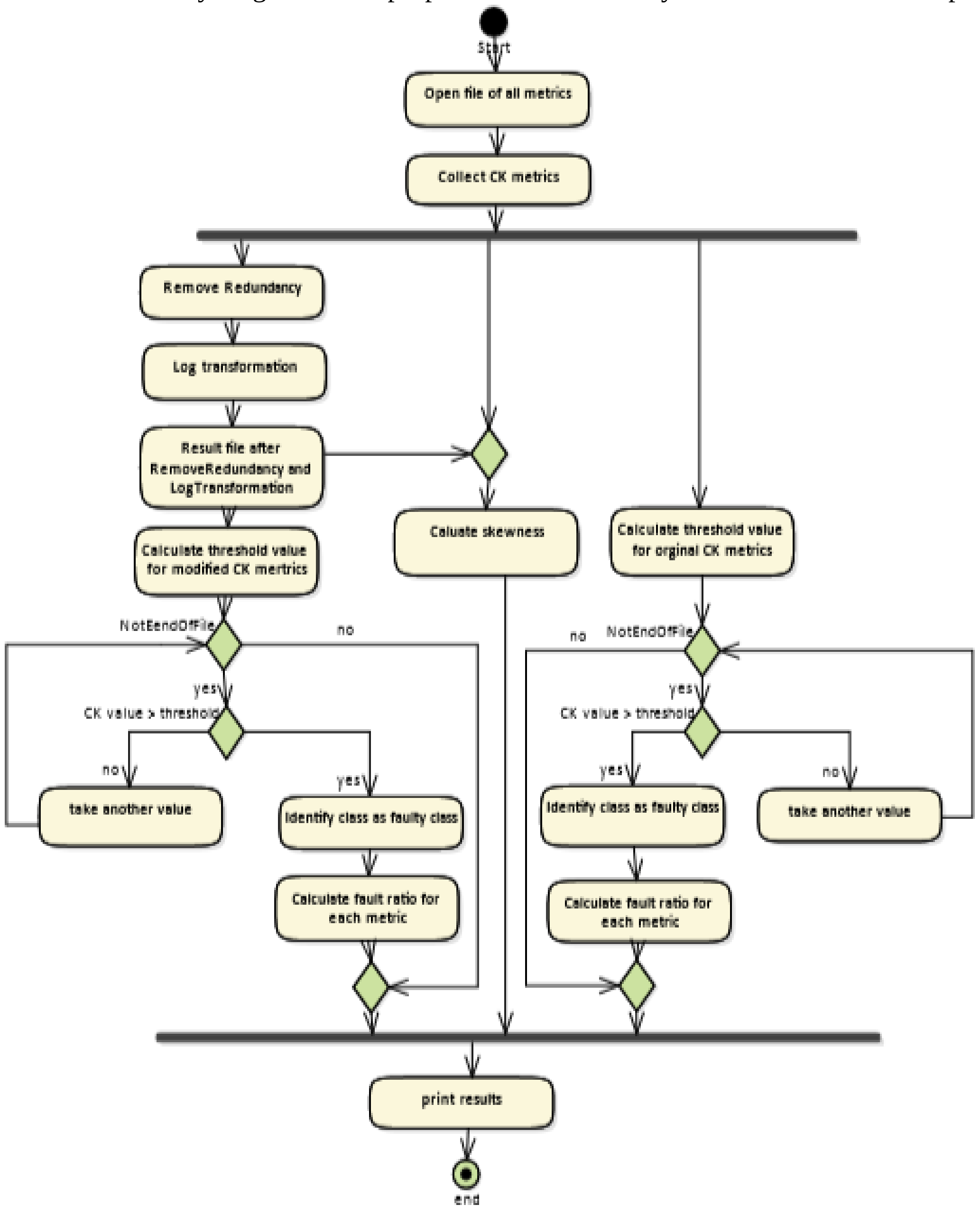
Start
Open file of all metrics
Collect CK metrics
Remove Redundancy
Log transformation
Result file after RemoveRedundancy and LogTransformation
Calculate threshold value for modified CK mertrics
Caluate skewness
Calculate threshold value for orginal CK metrics
NotEendOfFile
no
yes
CK value > threshold
no
yes
take another value
Identify class as faulty class
Calculate fault ratio for each metric
no
NotEndOfFile
yes
CK value > threshold
yes
no
Identify class as faulty class
take another value
Calculate fault ratio for each metric
print results
end

Figure 2- Activity diagram for the proposed model

IV. Case Studies

E-learning software [21] and system software [22] datasets consist of 64 and 65 classes, respectively. The datasets are publicly reported by Zenodo website.

- Practical steps of WMC metric before removing redundancy and before log transformation for e-learning software are calculated as follows:

Mean = Σ Xi /n =7.53571

Standard deviation= $\sqrt{(\Sigma (Xi - mean)^2 / n-1)}$ = 6.04733

Skewness= $\Sigma(Xi - mean)^3 / (n-1) * \sigma^3$ = 1.56767 where σ is the standard deviation.

Threshold = mean (M) + standard deviation (SD) = 13.58304

Faulty class= no. of classes > threshold value =9

Failure rate= 14.0625

- Practical steps of WMC metric after removing redundancy and after log transformation for e-learning software are calculated as follows:

Mean = Σ Xi /n =0.81087

Standard deviation= $\sqrt{(\Sigma (Xi - mean)^2 / n-1)}$ = 0.31989
Skewness= $\Sigma(Xi - mean)^3 / (n-1) *\sigma^3$ = 0.03611 where σ is the standard deviation
Threshold = mean (M) + standard deviation (SD) + 0.5 = 1.43076
Faulty class= no. of classes > threshold value =2
Failure rate= 3.125

The statistical results of other metrics are described in Tables- 2, 3, 4 and 5 of the case studies.

Table 2- Fault ratio of the file before removing redundancy and before log transformation for e-learning software.

Metric name	mean	Standard deviation	skewness	Threshold	Faulty class	Failure rate
WMC	7.53571	6.04733	1.56767	13.58304	9	14.0625
DIT	3.07813	1.67056	0.10518	4.74869	20	31.25
NOC	6.0	1.85164	0.15752	7.85164	1	1.5625
CBO	5.06452	4.15766	1.64654	9.22218	8	12.5
RFC	14.39286	12.74289	2.04199	27.13575	7	10.9375
LCOM	38.39024	61.63024	2.70416	100.02048	5	7.8125

Table 3- Fault ratio of the file after removing redundancy and after log transformation for e-learning software.

Metric name	mean	Standard deviation	skewness	Threshold	Faulty class	Failure rate
WMC	0.81087	0.31989	0.03611	1.43076	2	3.125
DIT	0.56137	0.15823	0.84264	1.01960	7	10.9375
NOC	0.75427	0.15005	0.75449	1.20433	1	1.5625
CBO	0.71971	0.27493	0.05102	1.29465	2	3.125
RFC	1.01856	0.38910	0.21735	1.70766	1	1.5625
LCOM	1.34576	0.57469	0.13142	2.22046	2	3.125

Table 4- Fault ratio of the file before removing redundancy and before log transformation for system software.

Metric name	mean	Standard deviation	skewness	Threshold	Faulty class	Failure rate
WMC	8.8	9.09641	2.83771	17.89641	7	10.76923
DIT	3.55385	6.94823	7.09210	10.50207	1	1.53846
NOC	0	0	0	0	0	0
CBO	7.31746	6.99166	3.20843	14.30912	5	7.69231
RFC	20.46154	20.25155	2.17235	40.71309	7	10.76923
LCOM	74.45833	180.62968	4.26238	255.08801	3	4.61538

Table 5- Fault ratio of the file after removing redundancy and after log transformation for system software.

Metric name	Mean	Standard deviation	Skewness	Threshold	Faulty class	Failure rate
WMC	0.83049	0.32953	0.5334	1.56001	3	4.61538
DIT	0.83652	0.39733	0.3778	1.63385	1	1.53846
NOC	0	0	0	0	0	0
CBO	0.84129	0.27076	0.44461	1.51206	3	4.6153846
RFC	1.13467	0.42181	0.23724	1.95648	1	1.53846
LCOM	1.43652	0.69733	0.3778	2.53385	2	3.07692

In all tables, the calculated value of the threshold is stated and, depending on it, the number of faulty classes and failure rate are computed. The results for E-learning and system datasets revealed that the fault ratio ranged from 1%-31% and 0%-10% for the original dataset and 1%-10% and 0%-4% after removing redundancy and log transformation. These results impacted directly the number of classes detected, which ranged between 1-20 and 1-7) for the original dataset and 1-7 and 0-3) after removing redundancy and log transformation.

As shown in Tables-3 and 5, the number of faulty classes and failure rate after removing redundancy and log transformation was decreased compared with those before removing redundancy and log transformation. Data in Tables- 2 and 4 are in the form that aids a software engineer to focus the attention on specific classes to improve the quality of these classes. The skewness of the dataset was also decreased.

V. Conclusions

In this research, a method for finding the fault rate of software was proposed depending on the object-oriented metrics. The paper concludes that:

- The proposed model classifies software classes as faulty or non-faulty based on metric values and the number of faulty classes, while the fault percentage is also determined.
- The paper employs two software as case studies. When the proposed model was applied, the accuracy of determining the number of faulty classes and failure rate in the datasets was increased after each operation (removal of redundancy and Log transformation)
- Decreasing the indicator of faulty classes and failure rate tends to reduce the effort of understanding the code. It also provides easy maintenance and easy refactoring by focusing on the classes that need more attention. Furthermore, it improves the quality and performance of the source code.

VI. Future Work

A set of improvements can be made, as summarized in the following points:

- Applying the proposed model on many datasets of software and comparing the results.
- Integrating the proposed model with other object-oriented metrics to predict faults of open source software to achieve more precise and reliable results.
- Expending the proposed model by using artificial intelligence algorithms.

References

1. Bird C., Nagappan N., Murphy B., Gall H. and Devanbu P. T. **2011**. “Don’t touch my code!: examining the effects of ownership on software quality”. In T. Gyimothy and A. Zeller, editors, SIGSOFT FSE, pages 4–14. ACM, 2011.
2. Okumura K., Okamura H. and Dohi T. **2018**. “Software Reliability Modeling and Analysis via Kernel-Based Approach”, 22nd International Conference on Engineering of Complex Computer Systems (ICECCS), Fukuoka, Japan, pp. 154-157.
3. Kumar L., Naik D. K. and Rath S. K. **2015**. Validating the Effectiveness of Object-Oriented Metrics for Predicting Maintainability, *Procedia Computer Science*, **57**: 798- 806.
4. Singha S.and Kaur S. **2018**. “A systematic literature review: Refactoring for disclosing code smellsin object oriented software”, *Ain Shams Engineering Journal*, **9**(4): 2129-2151.
5. Misra, S.; Adewumi, A., Fernandez-Sanz, L. and Damasevicius, R. **2018**. A Suite of Object Oriented Cognitive Complexity Metrics, *IEEE Access*, pp.8782–8796.
6. Nuñez-Varela A. S., Pérez-Gonzalez H. G., Martínez-Perez F. E. and Soubervielle-Montalvo C., **2017**. “Source code metrics: A systematic mapping study”, *Journal of Systems and Software*, **128**.
7. Manoj H.M and Nandakumar A.N, **2016**. “Constructing Relationship Between Software Metrics and Code Reusability in Object Oriented Design”, *International Journal of Advanced Computer Science and Applications*, **7**(2).
8. Bakar N. S. A. A. **2016**. ” *The Analysis of Object-Oriented Metrics in C++ Programs*”, Lecture Notes on Software Engineering, **4**(1), February 2016.
9. Shaik A., Reddy C. R. K., Manda B., Prakashini. C and Deepthi. K. **2010**. “Metrics for Object Oriented Design Software Systems: A Survey”, *Journal of Emerging Trends in Engineering and Applied Sciences (JETEAS)*, pp.190-198.

10. Ferreira K. A.M., Bigonha M. A.S., Bigonha R. S., Mendes L. F.O. and Almeida H. C. **2012**. "Identifying thresholds for object-oriented software metrics", *The Journal of Systems and Software*, pp. 244– 257.

11. Chawla M. K. and Chhabra I., **2012**. "Implementing Source Code Metrics for Software quality analysis", *International Journal of Engineering Research & Technology (IJERT)*, **1**(5).

12. Kapila H. and Singh S., **2013**. "Analysis of CK Metrics to predict Software Fault-Proneness using Bayesian Inference ", *International Journal of Computer Applications*, **74**(2).

13. Shatnawi R., **2015**. "Deriving metrics thresholds using log transformation", *Journal of software evolution and process*, pp.95–113.

14. Zhang F, Keivanloo I. and Zou Y. **2017**. "Data transformation in cross project defect prediction". *Empirical Software Engineering*., **22**(6): 3186-3218, DOI 10.1007/s10664-017-9516-2.

15. Gupta D. L. and Saxena K., **2017**. " Software bug prediction using object-oriented metrics ", Sadhana, *The Indian Academy of Sciences* , **42**(5): 655–669.

16. Gupta, Dharmendra Lal, and Kavita Saxena. **2017**. "*Software bug prediction using object-oriented metrics*." Sādhanā.

17. U P. and PK N. B. **2018**. Prediction of software defects using object- oriented metrics", *International Journal of Civil Engineering and Technology (IJCIET),* **9**(1): 889–899.

18. Zhu M. and Pham H., **2018**. " A software reliability model incorporating martingale process with gamma-distributed environmental factors", *Annals of Operations Research*, springer.

19. Belachew E. B., Gobena F. A. and Nigatu S. T., **2018**. "Analysis of software quality using software metrics" ,*International Journal on Computational Science & Applications (IJCSA)* **8**(4/5).

20. Rhmann W., Pandey B., Ansari G. and Pandey D.K. **2019**. "Software fault prediction based on change metrics using hybrid algorithms: An empirical study", *Journal of King Saud University – Computer and Information Sciences*, Elsevier.

21. https://zenodo.org/record/322433#.Xm6aGqgzbIU.

22. https://zenodo.org/record/322450.