Developing Software Bug Prediction Models Using Various Software Metrics as the Bug Indicators

Varuna Gupta
Research Scholar, Christ University, Bangalore

Dr. N. Ganeshan
Director, RICM, Bangalore

Dr. Tarun K. Singhal
Dean-Academic, INMANTEC, Gzb

***Abstract*—The bug prediction effectiveness reasonably contributes towards enhancing quality of software. Bug indicators contribute significantly in determining the bug prediction approaches and help in achieving software reliability. Various comparative research studies have indicated that Depth of Inheritance (DIT), Weighted Method per Class (WMC), Coupling between Objects (CBO) and Lines of Code (LoC) have significantly established themselves as reliable bug indicators for comprehensive bug predictions.**

The researchers have carried out a quantitative research and have developed prediction models using above bug indicators as models input and have applied these models on open source projects (Camel and Ant). During this research, the results demonstrates that there is significant correlation between size oriented metrics (bug indicators) such as DIT, WMC, CBO, LoC and bugs. Overall, DIT takes dominance in achieving better impact on predicting bugs than WMC, CBO and LoC.

The outcomes of the present research study would be of significance to software quality practitioners worldwide and would help them in prioritizing the efforts involved in bug prediction.

Keywords—Bug Prediction; DIT; WMC; CBO; LoC; SRGM

I. INTRODUCTION

Software reliability is considered critical and important aspect of software quality. Organizations pay due emphasis in detecting the quality of software product at an early stage to avoid late embarrassments arising due to late detection culminating in poor quality product ultimately. This approach ensures that organizations are able to redesign wherever possible and ensure consistent quality throughout. Organizations aim to ensure savings towards costs of development, reduction in time to develop and high reliability of software products.

Various attributes such as proneness to faults, testing efforts, maintenance efforts etc govern the quality of software products. Through this research, we have considered proneness to bugs as bug predictor utilizing DIT, WMC, CBO and LoC indicators within the realm of this research.

Various bug indicators proposed during last few decades have made the selection of right bug indicator a demanding task considering the complexity and nature of varying software development processes. In the wake, a number of researchers have predominantly proposed product oriented bug indicators. The testers across many organizations dedicate time and resources by allocating same priorities across all components of a project, which is not considered as an optimal approach.

Parts of the software systems don't have uniformity in bug distribution. This calls for comprehensive identification of files containing bugs throughout the project. The testers with such knowledge would be able to identify and prioritize the appropriate tests while achieving efficiency in testing process. In order to achieve the said, it is essential to ensure availability of appropriate software bug prediction models. The main objective of this research is to construct software bug prediction models using four bug indicators as the model input. The metrics collected by promise repository are used as the model input. Therefore, the model construction process allows assessment of appropriateness of the collected metrics as usable bug predictors. The predicted number of bugs for the files is the model output.

The present research has been organized into six sections. Section I introduces the concepts and practices being adopted in software bug prediction. Section II contains detailed review of literature. Section III demonstrates the process map adopted by the researchers. Section IV proposes modeling framework. Section V & VI contain analysis, conclusion and future research work.

Need of the Study

The generic realization is that software practitioners need to focus early on bug prediction approaches to ensure reasonable quality in software products. Therefore, a comprehensive research was needed to widen the scope of bug prediction approaches and identify bug indicators causing significant impact on software quality.

Objectives

1) To assess the correlation of bug indicators (DIT, WMC, CBO, LoC) with software bugs.

2) To develop software bug prediction models using bug indicators (DIT, WMC, CBO, LoC) as model inputs.

3) To compare the relative effectiveness of DIT, WMC, CBO and LoC towards prediction of bugs in Camel and Ant projects.

II. RELATED WORK

A significant amount of work has been cited using product metrics to predict bug prone files. Though major work has utilized Chidamber and Kemerer (CK) metrics suite [18] to

predict accurately pre and post release bugs in commercial and open source systems [23, 10, 8, 20, 13, 22]. Further, though CK metrics suite, empirical justification has also been made regarding usefulness in bug prediction [3, 6, 14].

Pareto analysis has also been used for evaluating the ability of models for identification of fault-prone classes, modules and files. As substantiated with presence of 80% of bugs in 20% of files [15, 26, 24, 25].

Linear regression has been widely considered as a common technique for bug prediction. Also DIT has been demonstrated to carry a linear relationship with bugs [16]. Further, our data was linear in nature advocating application of linear regression. Still, keeping with [1], which suggested application of nonlinear regression as better indicator for this type of data, so decided to go ahead with non linear regression.

Logistic regression models have also been used to identify fault-prone modules [4]. CK metrics suite was also used to find fault-prone classes [19]. This work involved investigation of two C++ written projects and followed with outcome involving analysis of 43-48% of classes to cover for 80% of the bugs

Bug prediction models were created based on the module size representing Line of Code (LoC). The models produced outputs in strong correlation with actual data [12]. These models suggested considering LoC in the bug prediction models.

A majority of CK metrics were found to be effective predictors for fault-proneness of class. In addition, DIT and Response for a Class (RFC) were found to be carrying more influence on the dependent variable [2].

A study on data from an industrial system comprising of more than 200 C++ subsystems added different metrics than CK metrics and applied logistic regression to evaluate those metrics. The outcomes suggested WMC and DIT as significant indicators for finding fault-prone classes [21].

Another research applying logistic regression on data from a telecommunication system having 174 C++ classes demonstrated close association of WMC, RFC and Coupling between Objects (CBO) with software bugs [5]. Another research using univariate logistic regression also identified WMC and SLOC as significant predictors [11].

Another research using data from two commercial applications, one having 150 classes and 23 KSLOC while other having 144 classes and 25 KSLOC evaluated the influence of six CK metrics on the number of bugs and identified RFC and DIT as most significant variables [19].

As per recent citations of the research works carried out, no significant amount of work has been done on the use of logistic reliability growth model for bug prediction.

Proneness to Bugs

Software failing to fulfill the specified requirement needs to be fixed. Signifying that the mistake has been committed between the initial requirement and the final operation of the software system. Since source code matters the most corresponding to the realization of the software system, the errors in source code are called bugs. There are changes that error may not become a bug. However, we need to fix it if it ultimately becomes a bug causing a failure. The proneness of bugs depends on reasons like DIT, WMC, CBO , LoC.,

DIT (Depth of Inheritance Tree): The maximum length from the root to a given class in the inheritance hierarchy. DIT is defined as the maximum length inheritance path from the class to the root class [19].

WMC (Weighted Methods per Class): WMC is defined as the sum of the complexity of the methods of the class. It is equal to the number of methods when all methods are of the complexity equal to UNITY. The sum of normalized complexity of every method in a given class.

CBO (Coupling Between Objects): The CBO metric represents the number of classes coupled to a given class. These couplings can occur through method calls, field accesses, inheritance, method arguments, return types and exceptions [18].

LOC (Line of Code): the LOC metric based on Java binary code represents sum of number of fields, number of methods and number of instructions in every method of the investigated class.

III. PROCESS MAP

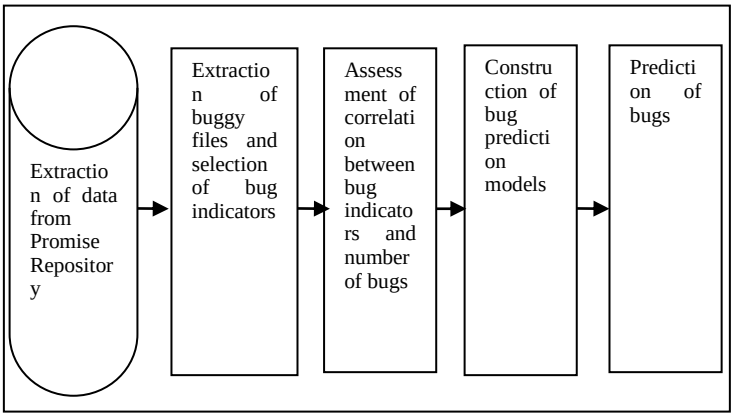

Fig. 1. Process Map

In this paper, the proposed process map is using the mixed method combining qualitative and quantitative research methods. The research work is detailed in five phases as shown in Figure 1.

A. Extraction of Data: Researchers have used PROMISE repository to extract the bug indicators (DIT, WMC, CBO and LoC) and bug data. The reason for selecting the open source projects from PROMISE repository was that it is a trustworthy software foundation having positive feedback from software users. It is also well-recognized in the software community.

B. *Extraction of buggy files and selection of bug indicators:* Two open source projects (Camel and Ant) were preferred to extract bug data from and selection of bug indicators for the analysis. Proper literature review was performed to select suitable bug indicators (DIT, WMC, CBO and LoC) for this research.

C. *Assessment of correlation between bug indicators and bugs:* Pearson correlation analysis was performed to assess the correlation between the various bug indicators (DIT, WMC, CBO, LoC) and number of bugs.

D. *Construction of prediction models:* After significant correlation between bug indicators and bugs, researchers have constructed prediction models using logistic software reliability growth model on extracted data from PROMISE bug database.

E. *Prediction:* After successful conclusion of the above four sub processes, finally predicted bugs was given as the model output.

IV. MODELLING FRAMWORK

A. Software Reliability Growth Models (SRGM)

Software reliability growth models are a statistical exclamation of detected bug's data using various mathematical functions. To predict the number of bugs in the code these mathematical functions are used. There are many types of software reliability growth models as to predict future bugs or failure rates.

B. Models Assumptions

Some of the general assumptions (apart from some special ones for specific models discussed) for the above model are as follows:

a) Software system is subject to failure during execution caused by bugs remaining in the system.

b) Failure rate of the software is equally affected by bugs remaining in the software.

c) The number of bugs predicted at any time instant is proportional to the actual number of bugs in the software.

d) Bug indicators referring the software size and its proportional impact on bugs have the capabilities of certain prediction.

e) All bugs are mutually independent from bug prediction point of view.

f)Bug prediction rate/bug detection rate is a logistic learning function as it is expected the learning process will grow with time.

g) The bug prediction phenomenon is modeled by Non Homogeneous Poisson Process (NHPP).

C. Models Notations

a- initial fault-content of the software.

k- A constant parameter in the logistic learning function

b_1- bug prediction rate/detection rate per unit time.

M (t) - expected number of bugs predicted.

Bug prediction models using SRGM are given by:

$$m(t) = a / \left(1 + k.e^{-b_1.t}\right) \qquad (4.1)$$

Prediction model-1

DIT is considered as a first model input referring to the below mentioned proposed model:

$$m(t) = a / \left(1 + k.e^{-b_1.dit}\right) \qquad (4.2)$$

Prediction model-2

WMC is defined as a second model input referring to the below mentioned proposed model:

$$m(t) = a / \left(1 + k.e^{-b_1.wmc}\right) \qquad (4.3)$$

Prediction model -3

CBO is defined as a third model input referring to the below mentioned proposed model:

$$m(t) = a / \left(1 + k.e^{-b_1.cbo}\right) \qquad (4.4)$$

Prediction model -4

LoC is defined as a fourth model input referring to the below mentioned proposed model:

$$m(t) = a / \left(1 + k.e^{-b_1.loc}\right) \qquad (4.5)$$

D. Goodness of Fit Criteria

The performance of a bug prediction model is judged by its ability to fit the past software reliability data and to predict satisfactorily the future behavior from present and past data behavior. The following criteria defined as:

1) *Coefficient of Multiple Determinations (R^2)*
2) *Bias*
3) *Variation*
4) *The Root Mean Square Prediction Error (RMSPE)*
5) *Mean Square Error (MSE)*

Bug Prediction Parameter Estimation

To examine the effectiveness of software bug prediction models using four indicators as model input, a set of comparison criteria is used to compare models quantitatively. The different comparison criterions used in our paper are as follows:

1) *Coefficient of Multiple Determination (R^2):*

This Goodness-of-fit measure has been used to investigate significance in trend existing in prediction of bugs. This coefficient was used as the ratio of the Sum of Squares (SS) derived from the trend model to that from a constant model subtracted from 1, that is

$$R^2 = 1 - \frac{residual \quad SS}{corrected \quad SS}$$

R^2 measures the percentage of the total variation about the mean accounted for by the fitted curve. It ranges in value from 0 to 1. Small values indicate that the model does not fit the data well. With movement of value towards 1, the model significantly explains the variation in the data [7].

2) Bias: The difference between the actual and predicted number of bugs at any instant of time i is known as Prediction Error (PE_i). The average of PEs is known as bias. With movement of value towards 0, the model significantly explains low presence of prediction error. The bias is defined mathematically as [9]:

$$Bias = \frac{\sum_{i=1}^{k}\left(\hat{m}(t_i) - m_i \right)}{k}$$

Where m_i indicates actual bugs, m(t) indicates predicted bugs and k is the number of observations in the data set.

3) Variance: The variance is defined as [9].

$$Variance = \sqrt{\frac{1}{k-1}\sum_{i=1}^{k}\left(m_i - \hat{m}(t_i) - Bias \right)^2}$$

4) Root Mean Square Prediction Error (RMSPE): It measures the closeness with which the model predicts the bugs and mathematical representation of this characteristic is given as [9].

$$RMSPE = \sqrt{Variance^2 + Bias^2}$$

5) Mean Square Error (MSE): MSE measures the difference between the predicted and actual values of bugs, and is given mathematically as **[**17**].**

$$MSE = \frac{\sum_{i=1}^{k}\left(m_i - \hat{m}(t_i) \right)^2}{k - p}$$

Where k is the number of observations in the data set and p is the number of parameters.

E. Data Sets

The data about bug indicators and bugs has been collected from PROMISE repositories. The following data sets have been used with explanations marked in:

Data Set 1(Camel) Apache Camel is a powerful open source integration framework based on known Enterprise Integration Patterns with powerful Bean Integration.

Data Set 2 (Ant) Ant is a well known Java-based, shell independent build tool.

V. Analysis and Conclusion

While checking the accuracy of different proposed models of bug prediction using different bug indicators, researchers have first estimated the unknown parameters of bug data for final software product on bug cumulative consumption data. Then, to judge the fitting of various proposed models of prediction given by equations (4.2), (4.3) (4.4) and (4.5) R^2, bias, variation, RMSPE and MSE have been calculated as the performance measures. Table I and Table II depict the estimated values for the parameters while Table III provides the correlation criteria and finally Table IV and Table V summarizes the estimated and optimized values of attributes of proposed models.

TABLE I. Estimated Parameters of Proposed Models Using Ds-1

S. No.	Parameters	Estimated parameters values			
		DIT	***WMC***	***LOC***	***CBO***
1	a	136.41	139.99	135.89	161.86
2	K	24.48	10.16	12.57	11.86
3	b_1	.071	.008	.001	.006

TABLE II. Estimated Parameters of Proposed Models Using Ds-2

S. No.	Parameters	Estimated parameters values			
		DIT	WMC	LOC	CBO
1	a	51.09	46.32	48.19	46.59
2	K	11.54	12.12	18.07	14.11
3	b_1	.046	.013	.001	.015

In our research, researchers observed significant correlations of WMC, DIT, CBO and LOC with bugs. In this research only highly correlated four metrics have shown from each data set that are listed in Table III. The interesting part of this result is that all four indicators are correlated significantly with software bugs.

TABLE III. Correlation Table

Project	Metrics	Correlation with Bugs
Camel	DIT	.976
	WMC	.987
	LOC	.984
	CBO	.992
Ant	DIT	.997
	WMC	.989
	LOC	.992
	CBO	.991

TABLE IV. Estimated and Optimal Values of Attributes for Four Prediction Models For Ds-1

Project	Metrics	R2	Bias	Variance	RMSE	MSE
Camel	DIT	99.5	-0.271	3.318	3.329	11.253
	WMC	98.9	0.183	4.712	4.716	23.048
	LOC	98.9	0.122	5.518	5.519	22.687
	CBO	98.6	0.141	5.271	5.273	28.88

TABLE V. Estimated and Optimal Values of Attributes for Four Prediction Models For Ds-2

Project	Metrics	R2	Bias	Variance	RMSE	MSE
Ant	DIT	99.1	0.089	1.349	1.352	1.893
	WMC	98.3	0.156	1.891	1.898	3.693
	LOC	98.9	0.132	1.505	1.511	2.333
	CBO	98.9	0.147	1.507	1.514	2.331

In table III researchers observed significant correlations of WMC, DIT, CBO and LOC with bugs. Table IV depicted that in case of DS-1 using prediction model 4.2 the predictive model coefficient of determination is 0.995 it means 99.5% of the variation in bugs is associated with number of predictor. Whereas using model 4.3, model 4.4 and model 4.5 the variation in bugs is 98.9%, 98.6% and 98.9% respectively.

Table V depicted that in case of DS-2 using prediction model 4.2 the predictive model coefficient of determination is 0.991 it means 99.1% of the variation in bugs is associated with number of predictor. Whereas using model 4.3 model 4.4 and model 4.5 the variation in bugs is 98.3%, 98.3% and 98.9% respectively.

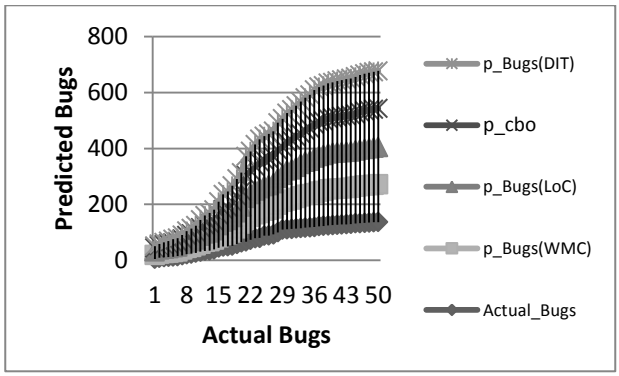

Fig. 2. Graph for Pattern of Actual and Predicted Software Bugs of DS-1

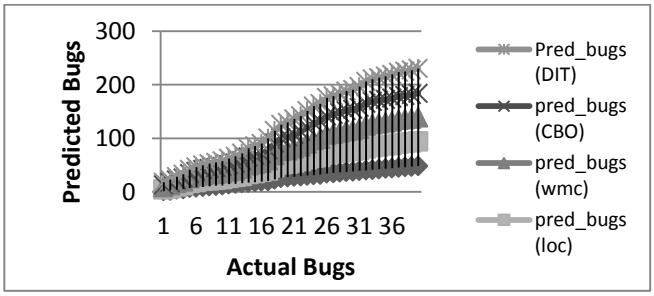

Fig. 3. Graph for Pattern of Actual and Predicted Software Bugs of DS-2

As shown above graphs in Figure – 2 and Figure – 3, the predicted number of bugs is significantly higher than actual number of bugs.

The research has comprehensively designed and tested four models using DIT, WMC, CBO and LoC as model inputs. These models produced significant results on all four model inputs. However, model using DIT as input was shown to be better performing than the other three models. This conclusion can serve as strong motivation for software practitioners to prioritize and allocate sufficient resources towards DIT because of its better performance in comparison to WMC, CBO and LoC.

VI. Future Work

More product metrics as bug indicators can be included in future research work. More open source data sets can also be included to bring higher reliability in bug prediction. An effort can be made of applying different non linear regression models on same two data sets already considered in present research work.

References

[1] A. Bernstein, J. Ekanayake, and M. Pinzger. Improving defect prediction using temporal features and non linear models. In Proc. Int'l Workshop on Principles of Softw. Evolution, pages 11–18, 2007.

[2] B asili , V.R., L.C. Briand and W.L. Melo, 1996. A validation of object-oriented design metrics as quality indicators. IEEE Trans. Software Eng., 22: 751-761. DOI10.1109/32.544352.

[3] CATAL C., DIRI B. and OZUMUT B., An Artificial Immune System Approach for Fault Prediction in Object-Oriented Software. Proc. of Dependability of Computer Systems, 2007, 238-245.

[4] DENARO G. and PEZZE M., An Empirical Evaluation of Fault-Proneness Models. Proc. of International Conference on Software Engineering (ICSE), 2002.

[5] El Emam, K., S. Benlarbi, N. Goel and S.N. Rai, 2001. The confounding effect of class size on the validity of object-oriented metrics. IEEE Trans. Software Eng., 27: 630-650. DOI: 10.1109/32.935855.

[6] JURECZKO M., Use of software metrics for finding weak points of object oriented projects. Proc. Of Metody i narzędzia wytwarzania oprogramowania 133-144, 2007 (in Polish).

[7] K. C. Chiu, Y. S. Huang, and T. Z. Lee, "A study of software reliability growth from the perspective of learning effects," Reliability Engineering and System Safety, pp. 1410–1421, 2008.

[8] K. E. Emam, W. Melo, and J. C. Machado. The prediction of faulty classes using object-oriented design metrics. J. Syst. Softw., 56(1):63–75, 2001.

[9] K. Pillai and V. S. S. Nair, "A model for software development effort and cost estimation," IEEE Trans. Softw. Engineering, vol. 23, no. 8, pp. 485–497, 1997

[10] L. C. Briand, J. W. Daly, and J. K. W¨ust. A unified framework for coupling measurement in object-oriented systems. IEEE Trans. Softw. Eng., 25(1):91– 121, 1999.

[11] Malhotra R., and Jain A., "Fault prediction using statistical and machine learning methods for improving software quality", Journal of Information Processing Systems, Vol.8, No.2, June 2012

[12] MENDE T., KOSCHKE R., Revisiting the Evaluation of Defect Prediction Models. Proc. of PROMISE, 2009.

[13] N. Nagappan, T. Ball, and A. Zeller. Mining metrics to predict component failures. In Proc. Int'l Conf. on Softw. Eng. (ICSE'06), pages 452–461, 2006.

[14] OLAGUE H. M., ETZKORN L. H., GHOLSTON S. and QUATTLEBAUM S., Empirical Validation of Three Software Metrics Suites to Predict Fault-Proneness of Object-Oriented Class Developed Using Highly Iterative or Agile Software Development Processes. IEEE Trans. on Software Engineering, 33(6), 2007, 402-419.

[15] OSTRAND T. J., WEYUKER E. J. and BELL R. M., Predicting the Location and Number of Faults in Large Software Systems. IEEE Trans. on Software Engineering, 31(4), 2005, 340-356.[8]

[16] S. Dowdy, S. Weardon, and D. Chilko. Statistics for Research. Probability and Statistics. JohnWiley and Sons, Hoboken, New Jersey, third edition, 2004.

[17] S. Hwang and H. Pham, "Quasi-renewal time-delay fault-removal consideration in software reliability modelling," IEEE Trans. Systems, Man and Cybernetics-Part A: Systems and Humans, vol. 39, no. 1, January 2009.

[18] S. R. Chidamber and C. F. Kemerer. A metrics suite for object oriented design. IEEE Trans. Softw. Eng., 20(6):476–493, 1994.[1]

[19] SUCCI G., PEDRYCZ W., STEFANOVIC M. and MILLER J., Practical assessment of the models for identification of defect-prone classes in object-oriented commercial systems using design metrics. Journal of Systems and Software 65(1), 2003, 1-12.

[20] T. Gyimothy, R. Ferenc, and I. Siket. Empirical validation of object-oriented metrics on open source software for fault prediction. IEEE Trans. Softw. Eng., 31(10):897–910, 2005.

[21] Tang, M.H., M.H. Kao and M.H. Chen, 1999. An empirical study on object-oriented metrics. Proceedings of the 6th International Symposium

on Software Metrics, Oct. 04-06, IEEE Computer Society, Boca Raton, FL., USA., pp: 242-249. DOI: 10.1109/METRIC.1999.809745.

[22] T. Zimmermann, R. Premraj, and A. Zeller. Predicting defects for Eclipse. In Proc. Int'l Workshop on Predictor Models in Software Engineering (PROMISE'07), pages 1–7, 2007.

[23] V. R. Basili, L. C. Briand, and W. L. Melo. A validation of object-oriented design metrics as quality indicators. IEEE Trans. Softw. Eng., 22(10):751–761, 1996.

[24] WEYUKER E. J., OSTRAND T. J. and BELL R. M., Adapting a Fault Prediction Model to Allow Widespread Usage. Proc. of PROMISE, 2006.

[25] WEYUKER E. J., OSTRAND T. J. and BELL R. M., Do too many cooks spoil the broth? Using the number of developers to enhance defect prediction models. Empirical Software Engineering, 13(5), 2008, 539-559.

[26] WEYUKER E. J., OSTRAND T. J. and BELL R. M., Comparing Negative Binomial and Recursive Partitioning Models for Fault Prediction. Proc. of PROMISE, 2008.