Empirical Analysis Of Open Source System For Predicting Smelly Classes

Harshpreet Kaur Saberwal[1]
Assistant Professor
Dept. of CSE
RIEIT, Railmajra.

Satwinder Singh[2]
Assistant Professor,
Dept. of CSE,
B.B.S.B.E.C, Fatehgarh Sahib.

Sarabjit Kaur[3]
Assistant Professor,
Dept. of CSE,
RIEIT, Railmajra.

Abstract

Software development plays an important role in software organization for high reliability and maintenance of the software. One such solution is refactoring that eases the code readability and maintainability. Refactoring is done by identifying bad smell areas in the code. An empirical model of object oriented software metrics is developed in this paper for prediction of bad smells. A binary statistical analysis presented here between metrics and bad smell which shows a significant relationship. Then a model is generated by using bad smell categorization. The proposed model is validated using dataset collected from jfreechart which shows that proposed model predict bad smell with high accuracy.

Keywords

Empirical Validation, Refactoring, Bad Smells, Statistical Methods.

1. Introduction

Software reengineering covers the entire software development process starting from requirement to testing and then beyond. The essence of software reengineering is to improve or transform existing software so that it can be understand and controlled. In the field of software engineering the design is the backbone of the software system and the software code supports it as a skeleton. If the skeleton is defective then the new changes may not be accommodated easily [1]. Software reengineering is important for recovering and reusing software assets. To accommodate complexity and iterative nature in software reengineering using object oriented framework, a new concept named, refactoring, has emerged in 1990s. The basic idea of refactoring is to clean up code in a controlled manner such that it minimizes the chances of introduction of bugs. Fowler and Beck [10] are the originators of bad smell design problems which help in improving the code quality of reengineered code. Bad smell is a hint that something has gone wrong somewhere in the code. Identifying bad smell in code helps to refactor the code. The author presented the problem in an informal essay style to guide a human developer manually to locate bugs within a system and provided a flat list of bad smells. But the author did not provided any precise criteria for evaluating code smells. The proposal is to avoid bad smell using refactoring. The main aim of refactoring for developing programs is to allow modifications without endangering external behaviour of the code. Fowler and Beck [10] describes a number of bad smells and referred refactoring to get rid of them. The author explained bad smell as a structure that needs to be removed from code by refactoring to improve the maintainability of the software but failed to suggest any criteria for making decisions regarding how to refactor code. After that Mantyla [12] extended the work on the empirical study of bad smell and evaluated the relationship between the bad smell and the software metrics of code. Later on Marticorena [13] studied the software metrics for the detection of bad smells. Those metrics can be used to find the bad smell within the code. Those Metrics may also be used to judge the quality of the software design. Some methods were proposed to improve the software quality and reduce the cost of testing and maintenance. As it is understood that the decision to refactor the code is not an easy, one a number of studies are carried out regarding decision making for refactoring and maintainability based on software metrics. In this present work the aim is to carry out a statistical study of relationship between six CK metrics and bad smell.

2. Literature Review

Significant work has been done in the field of bad smells. The review of those work covered in this paper. There are various categories of methods to predict bad smells such as statistical methods. The most common statistical methods used are univariate and multilogistic regression. Abreau et al. [1] has worked on the design quality of object-oriented software systems and evaluated the design attributes of object oriented software systems. These attributes can express the quality of internal structure of code. In this work, MOOD kit tool was applied on Object Oriented metrics for metrics extraction from source code (C++ code). Further the statistical theory was applied to MOOD metrics to evaluate the correlation between the sample value series. Fowler and Beck proposed metrics for removing bad smell design problems and presented the problem in an informal essay style to guide a human developer manually to locate bugs within a system. They provided a flat list of smells. The main aim was to achieve modifications without endangering external behaviour. But the work has not given any precise criteria for evaluating code smells. Briand et al. [8] have empirically investigated 49 metrics for predicting faulty classes. They use univariate and multivariate analysis to find the individual and combined effect of object oriented metrics and fault proneness. Mantyla et al. [18] proposed the taxonomy and initial empirical study of bad smell in code. This includes two contributions. The first is subjective taxonomy for the categorization of bad smell. This makes the smell more understandable than the single flat list of 22 bad smells. Secondly, the author provided the correlations between the smell. These correlations help in understanding the connection between different smells. Marticorena extended the taxonomy of bad smell with metrics. The extended metrics has been used for detecting bad smell and the metrics are granularity, coupling, inheritance, access and abstraction. Khomh and Penta proposed an exploratory study of the impact of code smell on software change proneness. The study showed that classes with smells are significantly more likely to be subject of changes than other classes. Singh et al. [20] had worked on the effectiveness of encapsulation and object oriented metrics to refactor code and identify error prone classes using bad smells. The work has been done by using the open source Firefox system. In this work the proper categorization of bad smell was generated for six CK metrics with additional metrics named public factor and encapsulation factor. This work has been done on C++ code by using Columbus Wrapper Framework. The work has established the relationship between bad smells and metrics. Regression analysis was used to analyse the results of the collected data. Then the univariate and Multinomial regression analyses were carried out to determine the relationship between the set of metrics. UBR and UMR analysis were used to shortlist the metrics on the basis of the significance of their association. Then find out the accuracy of the model using ROC curve.

3. Bad Smells in code

Identify bad smells in code helps to refactor the code. Refactoring of software code is a very tedious problem and applying it manually is yet more difficult. A number of surveys have been done for refactoring and maintainability. There is a need of external attributes to refactor the code for better understandability. Metrics provide solid information regarding object oriented properties. Research results show the relationship between structural attributes and external quality metrics. In this paper we find the association between bad smells and software metrics. In this a statistical analysis is carried out for each bad smell category. For this purpose we use the Analyst4j tool.

4. Research Methodology

In this we present the descriptive statistics for all the metrics that have considered.

4.1 Empirical Data Collection

This study makes use of jfreechart versions. Metrics and bad smell database was collected by use of Analyst4j tool. Each class is smelly if there is at least one bad smell is identified. After identifying bad smells categorize them into Table 1.

Table1. Bad Smell Categorization

SNo.	Bad Smell Category	Bad Smells
1.	Blob Class	• Large Objects • Large Attributes • Long Methods • Large Class • Long Parameter
2.	Undocumented Code	• No proper Documentation • Comments
3.	Using Inheritance	• Parallel Inheritance Hierarchies • Feature Envy
4.	Procedure oriented Design	• Switch Statements • Alternative classes with different interfaces
5.	Complex Class	• Duplicate Code • Data Class

4.2 Descriptive Analysis

Table 2 shows the mean, median, max, min of the independent variables by using descriptive analysis. Table 2 shows that the standard deviation of LCOM is low in jfreechart 1.0.0 pre1 because the value lies between 0 and 1. DIT also has low standard deviation in 1.0.1 version.

Table 2. Descriptive Statistics

Metrics	Jfreeechart 1.0.0 pre1				Jfreechart1.0.1			
	Mean	Std.Dev.	Min	Max	Mean	Std.Dev	Min	Max
LOC	129.87	198.626	1	1940	149.03	218.799	4	2209
WMC	22.51	43.819	0	495	23.13	45.574	0	528
RFC	46.21	80.121	0	820	48.65	82.236	0	847
LCOM	.33	.392	0	1	.68	6.199	0	135
CBO	8.71	9.245	0	70	9.90	9.557	0	69
DIT	1.39	1.097	0	21	1.41	.641	0	6

4.3 Method Used

Logistic regression is the commonly used statistical method. Logistic regression is used to predict the dependent variable from a set of independent variable [4,8]. It is used when outcome of the data input is binary. We use univariate binary and univariate multinomaial regression analysis. Univariate logistic regression used to find the relationship between dependent and each independent variable. It helps in finding the association between the metrics and bad smell. After that we choose the independent variables by passing them from multicollinearity test. Multicollinearity of metrics was removed by Variance Inflation Factor analysis. The limit of VIF is <10 and for tolerance it is > 0.1. After selecting the independent variables the empirical model was built by the logistic regression. In this two dependent variables were used to find the relation the relation between each type of metric and different levels of bad smell.

5. Result Analysis

In this section we have analyzed the results of our study. To start with the data analysis, first step is to collect the data set then apply statistical analysis. The statistical technique used for this purpose is univariate logistic regression. After identifying a subset of metrics we have to find the association between the metrics and bad smell.

5.1 Univariate Binary Analysis

In this univariate binary regression analysis is carried out to find the association between bad smell and metrics. In this we use the dependent binary variable and different CK metrics for finding the association between them. To shortlist the metrics we have to check the significance value of UBR analyses. Table 3. gives the UBR analysis showing the association between bad smells and metrics.

Table3. Univariate Binary Analysis

Metrics	Jfreechart 1.0.0 pre1		jfreechart1.0.1	
	B	p-value	B	p-value
LOC	.009	.000	0.008	.000
WMC	.034	.000	0.46	.000
RFC	.025	.000	0.027	.000
LCOM	N/A	N/A	N/A	N/A
CBO	.249	.000	0.207	.000
DIT	1.128	.000	0.703	.000

The metrics are associated with bad smell if the significance value p is less than 0.05. This UBR table shows that all the classes contain the bad smell except LCOM.

5.2 Univariate Multinomial Analysis

In univariate multinomial regression analysis the association was done on the basis of categories of bad smell. It can be seen from Table 4 that DIT and CBO is not associated with most of the categories of the bad smell. Rests of the metrics are helpful in predicting the bad smells in various classes.

Table4. Univariate Multinomial Regression Analysis

Metrics	Category	jfreechart1.0.0		jfreechart1.0.1	
		B	p-value	B	p-value
LOC	Undocumented Code	-.006	.000	-.004	.000
	Using Inheritance	-.007	.000	-.006	.000
	Procedure Oriented	.002	.001	.002	.008
	Complex Class	-.009	.000	-.005	.000
	Blob Classes	-.013	.000	-.005	.000
WMC	Undocumented Code	-.031	.000	-.015	.000
	Using Inheritance	-.053	.000	-.052	.000
	Procedure Oriented	.038	.000	.033	.008
	Complex Class	-.040	.000	-.021	.000
	Blob Classes	-.054	.000	-.023	.000
RFC	Undocumented Code	-.011	.000	-.007	.000
	Using Inheritance	-.015	.000	-.010	.000
	Procedure Oriented	.005	.003	.002	.000
	Complex Class	-.013	.000	-.008	.000
	Blob Classes	-.020	.000	-.009	.000
LCOM	Undocumented Code	-3.294	.000	-.001	.000
	Using Inheritance	-2.729	.000	.004	.000
	Procedure Oriented	7.470	.000	.747	**.121**
	Complex Class	3.781	.000	-.001	.000
	Blob Classes	-4.621	.000	-.003	.000
CBO	Undocumented Code	-.124	.000	-.089	.001
	Using Inheritance	-.086	.000	-.064	.050
	Procedure Oriented	-.012	**.217**	-.027	.003
	Complex Class	-.172	.000	-.121	**.976**
	Blob Classes	-.164	.000	-.096	**.920**
DIT	Undocumented Code	.693	.083	.860	.043
	Using Inheritance	.680	.000	1.557	.000
	Procedure Oriented	-2.722	.000	-3.290	.000
	Complex Class	.431	**.255**	.405	**.379**
	Blob Classes	.182	**.559**	.483	**.201**

Next choose the independent metrics by removing the collinearity from the metrics set.

Table5. Collinearity Statistics

Metrics	Before Dropping Metrics				After Dropping Metrics			
	Tolerance		VIF		Tolerance		VIF	
	Ver 1.0	Ver 1.1	Ver 1.0	Ver 1.1	Ver 1.0	Ver 1.1	Ver 1.0	Ver 1.1
LOC	.062	.069	16.140	14.505	N/A	N/A	N/A	N/A
WMC	.109	.123	9.203	8.155	.227	.246	4.407	4.605
RFC	.144	.151	6.967	6.626	.187	.209	5.336	4.791
LCOM	.782	.964	1.279	1.038	.785	.975	1.275	1.026
CBO	.306	.317	3.272	3.158	.347	.348	2.879	2.875
DIT	.950	.852	1.052	1.174	.856	.856	1.051	1.168

From Table5 it can be seen that the VIF value of LOC is greater than 10. So we have to exclude LOC because of high VIF. It is not helpful in predicting the bad smell. After excluding this we find out the likelihood ratio of the rest of the metrics. This likelihood ratio shows that the model is significant at 95% of confidence.

Table6. Model Fitness Test

	jfreechart1.0.0 pre1	Jfreechart 1.0.1
Likelihood Ratio	.000	.000

6. Conclusion

This paper proposes a method that categorizes similar bad smells. This helps in finding the association between the metrics and bad smells. An empirical study is carried out to find the association between the bad smells and the software metrics. In this paper we design a binary metrics model and then multivariate model to check the role of metrics in identifying the bad smell. We also find out the likelihood ratio to check the fitness of the model. This paper has described ongoing research on bad smells. This study was carried out on the open source system and the future work will extend to identify association between different projects depending on the language

7. References

[1] Abreau, F.B., M. Goulao, R. Esteves, Towards the design quality evaluation of object oriented software systems, Proc. 5th Int. Conf. on Software Quality, 1995.
[2]Abreau, F.B., Melo, Evaluating the impact of object oriented design on software quality, Proc. 3rd International Software mterics Symposium (Metrics96), IEEE, Berlin, Germany, March, 1996.
[3] Bansiya J, David CG, A hierarchical model for object oriented design quality, IEEE Transactions on software engineering, 2002, 28, pp, 4-17.
[4] Basili, V.L., Briand, L., Melo, W.L., A validation of object oriented metrics as quality indicators. IEEE Transations on software engineering, 1996, 22(10), pp. 751-761.
[5] Beck. K, Beedle M, van Bennekum A, Cockburn A, Cunninghum W, Fowler M, Grenning J et al. Manifesto for agile software development 2001.
[6] Briand, L., Arisholm, E., Counsell S., Houdek, F. and Thevenod-Fosse, p., Empirical Studies of OO Artifacts, Methods and Processes: State of Art and Future Direction, Empirical Software Engineering, 1999,4(4),387-404.
[7] Bieman, J., Kang, B.K., Measuring Design Level Cohesion, IEEE Transactions on Software Engineering, 1998,24(2), 111-124.
[8] Briand, L.c., Wuest, J., Daly, J.W., Porter, D.V., Exploring the relationship between design measures and software quality in OO systems, Journal of Systems and Software 2000, 5(3), 245-273.
[9] Cartwright, M., Shepperd, M., An empirical investigation of an object-oriented software system. IEEE transactions on Software Engineering, 2000, 26(7), 786-796.
[10] Chidamber S.R., Kermerer C.F., Towards a metrics suite for object oriented design, Proceedings of the Conference on Object-Oriented Programming: Systems,

Languages and Applications, 1991, 197-121.
[11] Chidamber, S.R., Kemerer, C.F., A Metric Suite for Object-Oriented Design, IEEE Transactions on Software Engineering, June 1994, 20(6), 476-493.
[12] Coleman D, Ash D, Lowther B, Oman PW, Using metrics to evaluate software system maintainability, IEEE Computing Practices, 1994, 27(8), 44-49.
[13] Emam, K.E., Melo, walcelio, Machado, Javam, The prediction of faulty classes using object oriented design metrics, The Journal of Systems and Software, 2001,56,63-75.
[14] F. Simon, F, Steinbruckner, F., Lewerentz, C., Metrics based refacoring. In Proceedings of the fifth European Conference on Software Maintenance and Reengineering, 2001.pp 30.
[15] Fawcett, T., ROC graphs: Notes and practical considerations for researchers Machine Learning, 2004,pp,31.
[16]Fowler, Martin, Refactoring: Improving the design of existing code. Addision-Wisely,2000.
[17]Grady RB, Successfully applying software metrics. IEEEComputer Vol 27, No. 9, pp. 18-25.
[18] Mika Mäntylä, Jari Vanhanen, and Casper Lassenius. 2003. A taxonomy and an initial empirical study of bad smells in code. In: Proceedings of the 19th International Conference on Software Maintenance (ICSM 2003). Amsterdam, The Netherlands. 22-26 September 2003, pages 381-384.
[19] Mika Mantyala. Bad Smells in Software- a Taxonomy and an empirical Study. PhD Thesis, Helsinki University of Technology, 2003.
[20]Satwinder Singh, K.S. Kahlon, Towards effectiveness of encapsulation and object-oriented metrics to refactor code and identify error prone classes using bad smells, Volume 36 number 5 2011.