ISSN No. 0976-5697

Volume 8, No. 5, May – June 2017

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

Prediction Model to Investigate Influence of Code Smells on Metrics in Apache Tomcat

Sharanpreet Kaur
Research Scholar
IKGPTU
Jalandhar, India

Dr. Satwinder Singh
Assistant Professor, Centre for Computer Science & Engg.
School of Engineering & Technology
Central University of Punjab Bathinda, India

Abstract: For advancing software maintenance process, attempts are necessitated at developers end. One such endeavour is applying refactoring to eliminate code smells from the software. The aim of refactoring process is to identify the smelly areas known as Code Smells. It makes the code livelier, easier to read and hence understanding of code increases. The aim of the paper is to perform an empirical analysis on the code smells and metrics. A set of object oriented metrics are selected for the study. Hence the study introduces a metric based prediction model of code smells. The paper initially introduces the statistical relationship between code smells and metrics. Based on the results neural network model development is made possible. The accuracy of the developed model is validated on machine learning algorithms. Four versions of Apache Tomcat (6.0, 7.0, 8.0, 8.5.11) are selected for the work. Successive versions of Tomcat source code are applied for validation of study. The results from the study revealed that metrics can predict smelly classes effectively

Keywords: code smells; refactoring; anti-pattern; open source software

I. INTRODUCTION

For achieving good quality of software product regular maintenance is necessary. Maintenance is performed in various forms like software testing, code inspections, walkthroughs and refactoring. Despite of these efforts faults still remain in the software. So the target of the developers should be at the root cause of faults. The cause for faults could be identified easily if the location where fault exists is examined. M. Fowler [1] had introduced a book on refactoring which revealed the problematic area in the code commonly known as "**Code *Smell***". The main objective of the paper is to identify such areas in the code and then apply metrics based heuristics to aid the maintenance process.

As object oriented metrics express internal quality attributes of software very effectively hence we consider them along with code smells. Software metrics are a great support to perform such type of empirical studies.

Object-oriented metrics [2] are used for the development of prediction model. We have used various object oriented metrics at package level; class level and method level which covers almost all concepts of object oriented programming such as encapsulation, inheritance, coupling, complexity and cohesion. The selected metrics are widely accepted in the literature by the researchers [3].

In this paper we have performed an empirical analysis on Apache Tomcat source code. It is a java based open source project. Tomcat is also called "Tomcat Server" which is a product of Apache Software Foundation (ASF) [4]. Tomcat is an implementation of Java Servlet and Java Server Pages (JSP) technologies. Four versions of Apache Tomcat (6.0, 7.0, 8.0, 8.5.11) are selected for the work.

We have performed the statistical analysis for developing the metrics based prediction model of code smells. Based on the results neural network model development is made possible. The accuracy of the developed model is validated on machine learning algorithms. Successive versions of Tomcat source code are applied for validation of study. The results from the study revealed that metrics can predict smelly classes effectively.

II. RELATED WORK

Some of the studies have been found in the literature aimed at performing refactoring which are given below.

Tom Mens et al. [5] revealed that refactoring could be possible at five different types. Five classes of source code i.e. Document, ASCIIDoc, PSDoc, PDFDoc, Printer, and Previewer are used to apply refactoring process. S. Counsell et al. [6] performed an empirical analysis to remove code smells from the source code of java source software. At least 21 code smells were being removed from the code.

Raed Shatnawi [6] tried to improve the quality of two open source projects - Eclipse & Struts using refactoring. Hierarchal approach had been used for quality management.

Seema Kansal [8] introduced new type of refactoring in order to remove code smells. Macro definitions had been put into the code where refactoring is demanded. Miryung Kim et al. [9] introduced pros and corns of refactoring at Microsoft. Three methods had been performed which involved a detailed investigation, structure consultations with software experts along with analysis of historical data. The study concluded that refactoring leads to minimizing risks and costs if faults are identified with time.

Davide Arcelli et al. [10] divulged that code performance could be achieved using Queuing Network Model of code smells and anti-patterns. Anshu Rani and Harpreet Kaur [11] performed an empirical investigation Intellij idea And Eclipse by comparing their results. Various refactoring characteristics have been studied along with refactoring tools.

Seyyed Ehsan Salamati Taba et al. [12] justified that relationship exists between faults and code smells. A fault prediction model has been proposed by considering the source code of open source projects- ArgoUML and Eclipse.

The study finalized that classes containing code smells are more prone to faults. Mesfin Abebe and Cheol-Jung Yoo [13] collected a huge data for performing refactoring from 1999 for applying refactoring.

M. Lakshmanan and S.Manikandan [14] revealed that identifying code smells in the source code is still a difficult task for developers. Although a wide range of detection tools are available. Yann-Gael Gueheneuc et al. [15] introduced a novel tool PTIDEJ for the detection of code smells, metrics and micro architectures. C++, Java code could be located easily for the design anomalies.

Sharanpreet Kaur and Satwinder Singh [16] performed a detailed systematic literature survey for detecting the categories of code smell detection. These techniques range from traditional to visualization based, semi automatic and automatic approach. Satwinder Singh and K.S .Kahlon [17] introduced a refactoring model which identifies code smells and its types in open source project Firefox. Relationship between OO metrics and code smells is justified based on results obtained. Two new types of metrics- PuF and EncF have been proposed. Satwinder Singh and Puneet Mittal [18] developed a model that discloses flaws in Firefox Mozilla based on metric value. High, Medium and Low category bugs are highlighted based on the approach. For generating the results from object oriented metrics tool named Columbus Wrapper Framework has been used.

III. DATA COLLECTION

The study involves the collection of four rereleases of Apache Tomcat version 6.0, 7.0, 8.0, 8.5.11. Only those releases of tomcat are shortlisted I which at least 5-7 % classes contains code smells. We have gathered information about code smells form Iplasma tool [19, 20, and 21] which is s a famous reverse engineering tool.

The metrics and code smells data is collected from tool. Selected metrics for the work empirical study are – ATFD, CBO, FDP, FANOUT, CC, CM, WOC, WMC, AMW, NOA, LOCC, DIT, HIT, NOD, NOAM, NOPA and TCC. Table I to IV represents the summarized metrics statistics results for Tomcat versions 6.0, 7.0, 8.0 and 8.5.11.

Code smells selected are- God Class, Data Class, Schizophrenic Class and Refused Parent Bequest. The metrics results are consolidated at class level. A short description about every code smell is given below.

a. **God Class** – It is a class which is enormously larger in size as compare to other classes in the system.
b. **Data Class** - The data class code smells reefers to a class which contains large amount of data while it is lacking in complex details.
c. **Schizophrenic Class** - It refers to a class in which many modifications are made for different reasons. It is also known as Divergent Change.
d. **Refused Parent Bequest** – This code smell is found where inheritance is applied in the source code. Such classes do not use the functionalities of the parent class.

.

Table I. Summarized Metrics Statistics of Tomcat Version 6.0

Tomcat 6.0						***Percentile***			
Quality Dimension	**Metric**	**Mean**	**Std Dev**	**Min**	**Max**	**25**	**50**	**75**	**90**
Coupling	ATFD	2.51	7.093	0	81	0.00	0.00	2.00	7.00
	CBO	3.32	6.109	0	53	0.00	1.00	4.00	10.00
	FDP	0.92	2.095	0	21	0.00	0.00	1.00	3.00
	FANOUT	9.82	21.627	0	206	0.00	1.00	9.00	29.00
	CC	2.35	10.763	0	286	0.00	0.00	1.00	5.00
	CM	6.77	40.990	0	1139	0.00	0.00	2.00	11.00
Complexity	WOC	0.6672	0.36377	0	1.00	0.3600	0.7800	1.0000	1.0000
	WMC	29.29	56.034	0	691	2.00	9.00	31.00	79.00
	AMW	1.9862	2.14934	0	29.00	1.0000	1.5000	2.5925	4.4970
	NOA	5.98	10.846	0	126	0.00	2.00	7.00	16.00
	LOCC	255.32	434.288	2	5962	40.00	109.50	286.25	649.70
Encapsulation	NOAM	3.02	8.034	0	102	0.00	0.00	3.00	8.00
	NOPA	0.40	1.892	0	36	0.00	0.00	0.00	1.00
Inheritance	DIT	0.63	0.929	0	5	0.00	0.00	1.00	2.00
	NOD	1.10	4.242	0	56	0.00	0.00	0.00	2.00
	HIT	0.31	0.688	0	4	0.00	0.00	0.00	1.00
Cohesion	TCC	0.3521	0.39162	0	1.00	0.0000	0.1900	0.6700	1.0000

Table II. Summarized Metrics Statistics of Tomcat Version 7.0

Tomcat 7.0						***Percentile***			
Quality Dimension	**Metric**	**Mean**	**Std Dev**	**Min**	**Max**	**25**	**50**	**75**	**90**
Coupling	ATFD	2.25	8.268	0	192	0.00	0.00	1.00	6.00
	CBO	3.28	6.394	0	63	0.00	1.00	4.00	9.00
	FDP	0.87	2.384	0	33	0.00	0.00	1.00	2.00
	FANOUT	8.82	20.645	0	251	0.00	1.00	8.00	25.00
	CC	2.25	11.327	0	289	0.00	0.00	1.00	4.00
	CM	6.08	42.458	0	1041	0.00	0.00	2.00	9.00
Complexity	WOC	0.6599	0.38552	0	1.00	0.3300	0.8300	1.0000	1.0000
	WMC	25.89	54.169	0	759	1.00	7.00	26.00	70.00
	AMW	1.9205	2.20182	0	36.00	0.8600	1.4000	2.5800	4.1760
	NOA	5.19	10.030	0	140	0.00	2.00	6.00	13.00
	LOCC	232.58	436.128	2	7099	31.00	93.00	235.00	576.00
Encapsulation	NOAM	2.47	7.820	0	136	0.00	0.00	2.00	6.00
	NOPA	0.21	1.432	0	36	0.00	0.00	0.00	0.00
Inheritance	DIT	0.71	1.070	0	5	0.00	0.00	1.00	2.00
	NOD	1.15	5.083	0	86	0.00	0.00	0.00	2.00
	HIT	0.33	0.703	0	5	0.00	0.00	0.00	1.00
Cohesion	TCC	0.3522	0.40163	0	1.00	0.0000	0.1700	0.7100	1.0000

Table III. Summarized Metrics Statistics of Tomcat Version 8.0

Tomcat 8.0						***Percentile***			
Quality Dimension	**Metric**	**Mean**	**Std Dev**	**Min**	**Max**	**25**	**50**	**75**	**90**
Coupling	ATFD	0.82	3.444	0	67	0.00	0.00	0.00	2.00
	CBO	1.12	2.699	0	44	0.00	0.00	1.00	3.00
	FDP	0.38	1.272	0	19	0.00	0.00	0.00	1.00
	FANOUT	2.51	6.744	0	72	0.00	0.00	2.00	7.00
	CC	0.69	4.320	0	128	0.00	0.00	0.00	1.00
	CM	1.37	9.383	0	272	0.00	0.00	0.00	2.00
Complexity	WOC	0.6533	0.41224	0	1.00	0.2200	0.9300	1.0000	1.0000
	WMC	15.67	42.819	0	827	0.00	4.00	14.00	37.00
	AMW	1.5658	1.87144	0	15.50	0.0000	1.0000	2.0000	3.7710
	NOA	3.43	7.398	0	126	0.00	1.00	4.00	8.00
	LOCC	134.83	224.532	2	3139	22.00	62.00	146.00	330.40
Encapsulation	NOAM	1.56	5.794	0	140	0.00	0.00	1.00	4.00
	NOPA	0.15	1.455	0	36	0.00	0.00	0.00	0.00
Inheritance	DIT	0.62	0.950	0	5	0.00	0.00	1.00	2.00
	NOD	0.84	3.409	0	41	0.00	0.00	0.00	2.00
	HIT	0.28	0.658	0	5	0.00	0.00	0.00	1.00
Cohesion	TCC	0.2962	0.41613	0	1.00	0.0000	0.0000	0.6725	1.0000

Table IV. Summarized Metrics Statistics of Tomcat Version 8.5.11

Tomcat 8.5.11						***Percentile***			
Quality Dimension	**Metric**	**Mean**	**Std Dev**	**Min**	**Max**	**25**	**50**	**75**	**90**
Coupling	ATFD	0.88	4.105	0	66	0.00	0.00	0.00	2.00
	CBO	1.09	2.838	0	44	0.00	0.00	1.00	3.00
	FDP	0.39	1.357	0	19	0.00	0.00	0.00	1.00
	FANOUT	2.50	7.853	0	122	0.00	0.00	1.00	7.00
	CC	0.68	4.173	0	122	0.00	0.00	0.00	1.00
	CM	1.43	9.369	0	267	0.00	0.00	0.00	2.00
Complexity	WOC	0.6555	0.41370	0	1.00	0.2175	0.9600	1.0000	1.0000
	WMC	15.21	43.188	0	827	0.00	4.00	12.25	34.90
	AMW	1.5022	1.79991	0	15.50	0.0000	1.0000	2.0000	3.6970
	NOA	3.39	7.653	0	142	0.00	1.00	4.00	8.00
	LOCC	131.73	232.928	2	3139	21.00	58.00	137.00	317.70
Encapsulation	NOAM	1.53	5.756	0	140	0.00	0.00	1.00	4.00
	NOPA	0.15	1.456	0	36	0.00	0.00	0.00	0.00
Inheritance	DIT	0.61	0.960	0	5	0.00	0.00	1.00	2.00
	NOD	0.80	3.051	0	39	0.00	0.00	0.00	2.00
	HIT	0.29	0.638	0	4	0.00	0.00	0.00	1.00
Cohesion	TCC	0.2933	0.41699	0	1.00	0.0000	0.0000	0.6725	1.0000

Table V. UMR Analysis of Tomcat Versions

UMR Test	***Tomcat 6.0***		***Tomcat 7.0***		***Tomcat 8.0***		***Tomcat 8.5.11***	
	B	**Sig.**	**B**	**Sig.**	**B**	**Sig.**	**B**	**Sig.**
ATFD	-.035	**.861**	.095	**.000**	.337	**.000**	.075	**.794**
CBO	-.073	**.000**	.031	**.740**	.334	**.000**	-.228	**.000**
DIT	-.852	**.000**	-1.047	**.000**	-1.235	**.000**	-1.537	**.000**
HIT	-.243	.677	.011	.962	.003	.993	.027	.956
LOCC	-.009	**.000**	-.003	**.000**	-.011	**.000**	-.005	**.000**
NOA	-.033	.000	.038	.487	.009	.711	.008	.737
NOD	.005	.949	.005	.874	.000	.999	1.211	1.000
NOM	.584	**.000**	.101	**.000**	.273	**.000**	.224	**.000**
WMC	-.015	**.000**	-.015	**.000**	.127	**.001**	.061	**.000**
AMW	-.073	**.540**	-.099	**.000**	-.368	**.007**	-.260	**.000**
FANOUT	.009	**.389**	-.085	**.000**	-.159	**.000**	-.004	**.914**
WOC	.247	.533	3.980	**.000**	5.333	**.000**	4.066	**.000**
CC	-.834	**.000**	-.079	.298	-11.35	.995	.861	**.000**
CM	.202	**.000**	-.049	.718	-.105	.000	-.122	**.000**
NOAM	-.807	**.000**	-.178	**.000**	-.633	**.000**	-.462	**.000**
NOPA	-.660	**.000**	-.324	**.000**	-.684	**.000**	-.518	**.000**

In the empirical analysis we have used logistic regression for code smell prediction based on metrics value. Initially Regression Analysis is applied to examine the relationship among variables. We have applied Univariate Multinominal Regression (UMR) test to evaluate the relationship between metrics and code smells based upon the cut off p-value (.05). Metrics are selected as independent variables for the study. The independent variables are removed from the analysis using PCA (Principle Component Analysis). For the development of prediction model Neural Network MLP is applied which is a powerful tool support for prediction. The accuracy of predicted model is confirmed by testing model successive versions. The area under Receivable Operating Characteristics (ROC) Curve is used to evaluate the accuracy of the model. Testing of model is verified with machine learning algorithms.

IV. RESULTS

In this section we present the results obtained from the mentioned datasets and their concluded outputs obtained.

IV a. UNIVARIATE MULTINOMINAL REGRESSION

We have performed Univariate Multinominal Regression (UMR) upon the code smells and the metrics results obtained. Table V depicts the results after UMR Test. *P-value* is examined and metrics values less than .05 are omitted from the study. All the selected metrics p-value is less than .05 except few metrics like HIT, NOA and NOD. So we are omitting HIT, NOA and NOD metrics.
Similarly CC and CM metrics are fulfilling the *p value* for only two versions of Tomcat. So these are also omitted. TCC metric is excluded from the study after PCA. Hence the final set of metrics is - ATFD, CBO, FDP, FANOUT, WOC, WMC, AMW, LOCC, DIT, NOAM, and NOPA which is used for prediction model development.

IV b. NEURAL NETWORK

Neural Network Model MLP (Multi Layer Perception) is used for the development of prediction model. Hyperbolic Tangent Function and Softmax Activation Function are applied for Hidden Layer and Output Layer respectively. We have calculated the area under ROC curve for the evaluation of model. The ROC curve plots the probability between true positives and false positives for the curve range between 0 and 1. The range of the discrimination is as follows:

- 0.5 <= ROC < 0.6 It means no discrimination
- 0.6 <= ROC < 0.7 It means poor discrimination
- 0.7 <= ROC < 0.8 It means good discrimination
- 0.8 <= ROC < 0.9 It means excellent discrimination
- 0.9 <= ROC < 1 It means outstanding discrimination

The NN MLP area under ROC curve is represented in Table VI which is given below. The model generates an excellent discrimination for the results

Table VI. Area under ROC curve for NN model

Tomcat Versions	*Class Level*
Tomcat 6.0	.871
Tomcat 7.0	.852
Tomcat 8.0	.853
Tomcat 8.5.11	.861

IV c. EVALUATING MODELS ON NEXT RELEASES

The aim of the paper is to develop the model to predict code smells based on metrics value. For the validation of work successive models are applied on the releases of Tomcat. We have applied Tomcat 6.0 on Tomcat 7.0, 8.0 and 8.5.11 and so on. Table VII reveals the results.

Table VII. Testing of Code smells prediction model

Applications of Model	***Tomcat 7.0***	***Tomcat 8.0***	***Tomcat 8.5.11***
Applying Tomcat 6.0	0.981	0.921	0.947
Applying Tomcat 7.0	-	0.972	0.965
Applying Tomcat 8.0	-	-	0.987

V. CONCLUSION

We have tried to develop the metric based code smells prediction model. Although from the set of selected metrics all are not fulfilling the p-value in the UMR Analysis. So we have used the metrics accomplishing the task of model development. A good accuracy has been shown by the ROC curve. So we conclude that the predicted model can work satisfactory in general.

VI. REFERENCES

[1]. Martin Fowler, Martin Beck, Kent Brant, John Opdyke, William Roberts, “Refactoring- improving the design of existing code” 1st Ed. Addison-Wesley, June 1999.

[2]. Szabo, R. and Khoshgoftaar, T. “An assessment of software quality in a C++ environment”, Proceedings of the 6th Int. Symposium on Software Reliability Engineering, pp.240–249 1995.

[3]. Francesca Arcelli Fontana, Mika V. Mantyla, Marco Zanoni and Alessandro Marino, “Comparing and experimenting machine learning techniques for code smell detection”, Empirical Software Engineering pp 1143–1191 2015.

[4]. Online Sources: http://tomcat.apache.org/.

[5]. Tom Mens and Tom Tourw, “A Survey of Software Refactoring”, IEEE Transactions On Software Engg. Vol. 30, No. 2, February 2004

[6]. S. Counsell, R.M. Hierons, H. Hamza, S. Black and M. Durrand, “Exploring the Eradication of Code Smells: An Empirical and Theoretical Perspective”, Advances in Software Engineering Volume 2010, Article ID 820103, pp 1-12

[7]. Raed Shatnawi, “An Empirical Assessment of Refactoring Impact on Software Quality Using a Hierarchical Quality Model”, International Journal of Software Engineering and Its Applications Vol. 5 No. 4, October, 2011 127

[8]. Seema Kansal, “Refactor Code: A Review” IJCST Vol. 2, Issue 2 June2011.

[9].Miryung Kim, Thomas Zimmermann and Nachiappan Nagappan, “Field Study of Refactoring Challenges and Benefits” SIGSOFT/FSE’12 November 2012, Raleigh, NC, USA Copyright 2012 ACM.

[10]. Davide Arcelli, Vittorio Cortellessa,Catia Trubiani, “Antipattern-Based Model Refactoring for Software Performance Improvement” , ACM 2012.

[11]. Anshu Rani and Harpreet Kaur, “Refactoring Methods and Tools” International Journal of Advanced Research in Computer Science and Software Engineering Research Volume 2, Issue 12, December 2012.

[12]. Seyyed Ehsan Salamati Taba,Foutse Khomh ,Ying Zou, Ahmed E. Hassan, and Meiyappan Nagappan, “Predicting Bugs Using

Antipatterns" ICSM '13 Proceedings of the IEEE International Conference on Software Maintenance pp 270-279 2013.

[13]. Mesfin Abebe and Cheol-Jung Yoo Chonbuk, "Trends, Opportunities and Challenges of Software Refactoring: A Systematic Literature Review" International Journal of Software Engineering and Its Applications Vol.8, No.6 (2014), pp.299-318.

[14]. M.Lakshmanan, S.Manikandan, "Multi-Step Automated Refactoring For Code Smell" IJRET: International Journal of Research in Engineering and Technology, Volume: 03 Issue: 03 | Mar-2014.

[15]. Yann Gael Gueheneuc, Herve Albin-Amiot and Ecole des Mines de Nantes, "Using Design Patterns and Constraints to Automate the Detection and Correction of Inter-class Design Defects", Paper accepted at TOOLS USA 2001.

[16]. Sharanpreet Kaur and Satwinder Singh, "Influence of Anti-Patterns on Software Maintenance: A Review", International Journal of Computer Applications, International Conference on Advancements in Engineering and Technology (ICAET 2015) pp 14-19 2015.

[17]. Satwinder Singh and K.S .Kahlon, "Effectiveness of Refactoring Metrics Model to Identify Smelly and Error Prone Classes in Open Source Software", ACM SIGSOFT Software Engineering Notes Page 1 March 2012 Volume 37 Number 2 pp 1-11.

[18]. Satwinder Singh and Puneet Mittal, "Empirical model for predicting high , medium and low severity faults using object oriented metrics in Mozilla Firefox", Int. J. Computer Applications in Technology, Vol. 47, Nos. 2/3, 2013 pp 110-124.

[19]. R.Marinescu, "Detection Strategies: Metrics-Based Rules for Detecting Design Flaws", In: Proc. 20th Int. Conf. Software Maintenance, pp. 350-359, 2004.

[20]. Francesca Arcelli Fontana, Vincenzo Ferme, Alessandro Marino, Bartosz Walter, Pawel Martenka, "Investigating the Impact of Code Smells on System's Quality: An Empirical Study on Systems of Different Application Domains", 2013 IEEE International Conference on Software Maintenance pp 260-269.

[21]. Francesca Arcelli Fontana, Elia Mariani , Andrea Morniroli, Raul Sormani, Alberto Tonello, "An experience report on using code smells detection tools", Proceedings - 4th IEEE International Conference on Software Testing, Verification, and Validation Workshops, ICSTW 2011, pp 450-457