International Journal of Current Engineering and Technology
E-ISSN 2277 – 4106, P-ISSN 2347 - 5161
©2014 INPRESSCO®, All Rights Reserved
Available at http://inpressco.com/category/ijcet

Research Article

Bayesian Inference to Predict Smelly classes Probability in Open source software

Heena Kapila[A*] and Satwinder Singh[B]

[A]Department of Information Technology, Chandigarh Engineering College, Landran-140307, India
[B]Department of Computer Science & InfoTech ,B.B.S.B.E.C, Fatehgarh Sahib-140407, India

Accepted 15 May 2014, Available online 01 June2014, **Vol.4, No.3 (June 2014)**

Abstract

Software testing entails a number of processes that are focused on finding faults within a stipulated time. Lots of papers have been published for object oriented metrics but mostly concentrating on software fault prediction, very few has been published for bad smells. Bad code smells are used to recognize complex classes in object-oriented software systems for refactoring. This study contributes to all code smell prediction techniques by designing a Logistic regression model and using Bayesian inference graphs. This paper shows the results of a study in which Object Oriented metrics effectively predict design smell for an open source system. Software metrics assess as predictor of smelly classes. Bayesian inference graphs can represent decision for finding the smells present in software system. For Probabilistic reliability analysis, Bayesian inference is intended to be used for risk related data. This paper presents the relationship between smelly classes and object-oriented metrics. This study demonstrates a statistical technique for estimating the smelly classes for any piece of software. We examined the open source Eclipse system, which has a strong industrial usage. Our main objective is to design a Bayesian Inference graph to predict bad smell in the code.

Keywords: *Bayesian Inference, Smelly classes, Software Reliability, CK Metrics, Logistic Regression, Object Oriented Metrics*

1. Introduction

There are many areas in which Software has become an essential part of many real time systems, including medical, power plant and air traffic control applications (Shatnawi, 2010). It is very difficult to develop such software applications because system engineers have to deal with a large number of quality requirements, dependability and performance. The human dependency on software gives rise to the possibilities of crises from its failure (Shatnawi, 2010). Therefore, there is an increasing need for fault free software systems. In object oriented languages, software metrics has been used for many years to provide developer with additional piece of information about their programs (Singh *et. al*, 2011). The establishment of software reliability processes to recognize software faults before release into the environment is crucial. Software fault analysis at later stages of system development increase software corrective maintenance cost (Dejager *et. al*, 2013). The number of faults in a software system can be very large and Software maintenance expenses increase when faults are detected in the later stages of the software development life cycle (Dejager *et. al*, 2013). The Software industry is paying more attention to detecting errors in software systems, which are very common and complex problems. Preventing a software system from having errors is a difficult task (Bennett *et. al*, 2000). For reducing the defect rate and increase the effectiveness, it is important to identify smelly modules in software (Kapila *et. al* 2013). The main focus of this essay is to find a association between smelly modules and software metrics. A software engineer should always plan for changes to make software more reliable and free from bad code smell.

To reduce the risk of smelly classes, the developer should choose the best method for design (Kapila *et. al* , 2013). This study represents the result of empirically investigated the CK metric suite and proposed model can establish a relationship between software metrics and code smell. Prediction of smelly classes provides a method to maintain software quality engineering (Shatnawi *et. al*, 2010). The quality of software is extremely important. For improving quality, developers should focus on testing those portions of code that have the largest number of smelly classes. This study is an attempt to predict bad code smell in software by designing a Logistic regression model and using Bayesian inference graphs.

Software quality is a necessary issue throughout the software industry. Early estimating the smelly classes of software could assist on improving software quality. Prediction of smelly class is a proven technique for attaining high software reliability (Kapila *et. al*, 2013). Jiantao Pan stated that "Software reliability is the probability of failure-free software operation for a specified period of time in a specified environment." Techniques for estimating the testing attempts can help in increasing the effectiveness of software testing (Kaur *et.*

*Corresponding author: **Heena Kapila**

al, 2011). Being able to predict the smelly classes of software can help in improving the efficiency of the whole process. The main problems with many software systems are that they have bad code smells. For preventing software system from bad code smell, software developers must know where errors are likely to occur. Many researchers studied and designed many metrics models (Rosenberg, 1998).

Software metrics can be used to predict which modules that are expected to contain code smells. Software fault prediction is a topic of main concern for various researchers (Kapila *et. al*, 2013) . A timely identification of smelly classes will allow for a more effective allotment of testing resources and enhanced software quality (Dejaeger *et. al* 2013). The increasing need for software-based systems raise the need to develop high quality and reliable software system. Identifying software faults at early stages of system deployment can decrease software maintenance costs and increase software quality. Software quality estimation is not only concerned about reliability, but also the other quality characteristics such as usability, efficiency, maintainability, functionality, and portability. Software maintenance is defined in IEEE Standard 1219 (Bennett *et. al*, 2000) as "The modification of a software product after delivery to correct faults, to improve performance or other attributes, or to adapt the product to a modified environment."

In this study Logistic regression and Bayesian Inference are used to determine the probabilistic influence relationships between software metrics and smelly classes. Logistic regression is a statistical technique for predicting the probability. Bayesian Inference is used in works concerning maintenance decisions and performance evaluation (Weber, *et. al* 2012). Lots of different metrics can be used for predicting defects in software. It would be more beneficial to deal with fewer more-important metrics, rather than deal with more less-important metrics. These metrics were extracted by inspecting the source code of Eclipse 3.4.

2. Related Literature

It has been proven that the division of faults over a system can be represented by a Weibull probability distribution. In Weibull analysis, the method to provide logically accurate failure analysis is done forecasts with small samples. According to Dejaeger *et. al* (2003) "To construct a prediction model, which discriminates between fault-prone code segments and those presumed to be fault-free, the use of static code features in code segments has been constructed". (Bender 1999) proposed a method for quantitative risk assessment in epidemiological studies. (Bender 1999) investigated threshold effects between a binary responsible variable and a continuous risk factor. According to Bender "The corresponding benchmark values of the risk factor can be calculated by defining acceptable levels for the absolute risk and the risk gradient. These values can be calculated by means of nonlinear functions of the logistic regression coefficients." Bender's proposed method is illustrated by applying it on medical patient data to find out the value of an Acceptable Risk Level (VARL) and the Value of an Acceptable Risk Gradient (VARG). These values are used to predict probability risk levels. Defect counts work as the indicator of the quality of the OO system, allowing for more variation in the dependent variables analyzed by Subramanyam *et. al* (2003). They validated the WMC (Weighted Methods per Class), CBO (Coupling Between Object classes), and DIT (Depth of Inheritance Tree) metrics as parameters to predict the error count in a class. According subramanyam CK metrics could predict error counts. Subramanyam and Krishnan select a huge e-commerce system developed in C++ and Java. They empirically validated the CK suite of OO design metrics for both languages, C++ and Java. They measured the consequences of size along with the metrics values on the number of faults by using regression analysis. They represented that WMC and CBO could be validated only for C++. Alshayeb and Li(2003) proposed a study to represent the association between object oriented metrics. They studied two client–server systems and measured changes in three Java Development Kit (JDK) releases. According to Alshayeb and Li(2003) "They found that OO metrics are effective in predicting design efforts and source lines of code added, changed, and deleted in the short-cycled agile process and ineffective in predicting the same aspects in the long-cycled framework process." They found that the Object Oriented metrics effectively predicted design effort. However, the metrics were not efficient predictors of some variables in the long-cycled framework evolution process (JDK). TiborGyimo *et. al* (2005) illustrated fault-proneness prediction of the source code of the open source system. They used two different methods for fault-proneness prediction. Regression and machine learning methods are used to authenticate these metrics. They collected and checked the metrics value against the bugs found in the bug database called Bugzilla. They found CBO (Coupling Between Object classes) and LOC (Line of Code) metrics to be the best in predicting the fault-proneness of classes but the DIT metric is untrustworthy and NOC cannot be used at all for fault-proneness prediction. (Catal *et. al* 2009) proposed a method that does not require a human expert during the prediction process. According to (Catal *et. al* 2009) "Experiments reveal that unsupervised software fault prediction can be automated and reasonable results can be produced with techniques based on metrics thresholds and clustering."

(Shatnawi, 2000) proposed threshold values with enhanced classification accuracy. Shatnawi stated that "Threshold values provide a meaningful interpretation for metrics and provide a surrogate to identify classes at risk." The classes that exceed a threshold value can be selected for more testing. In this study Shatnawi measured the efficiency of a statistical methodology to recognize threshold values for the OO metrics. This methodology can be used to recognize threshold values based on the logistic regression model. (Shatnawi *et. al*, 2010) tested threshold values of software metrics in binary and the ordinal category by using ROC (Receiver Operating Characteristic) curve analysis. They used the area under the curve to choose threshold values for each metric.

Various approaches for software defect predictions are reviewed by Fenton and Neil (Fenton *et. al*, 1999). This study concluded that traditional statistical techniques, such as using regression modelling alone, were insufficient.

3. Experimental Design

This study has two objectives.

1. The first object is to find probabilistic assumptions based on Logistic regression.
2. The second objective is to predict the probability of occurrences or non occurrence of smelly classes.

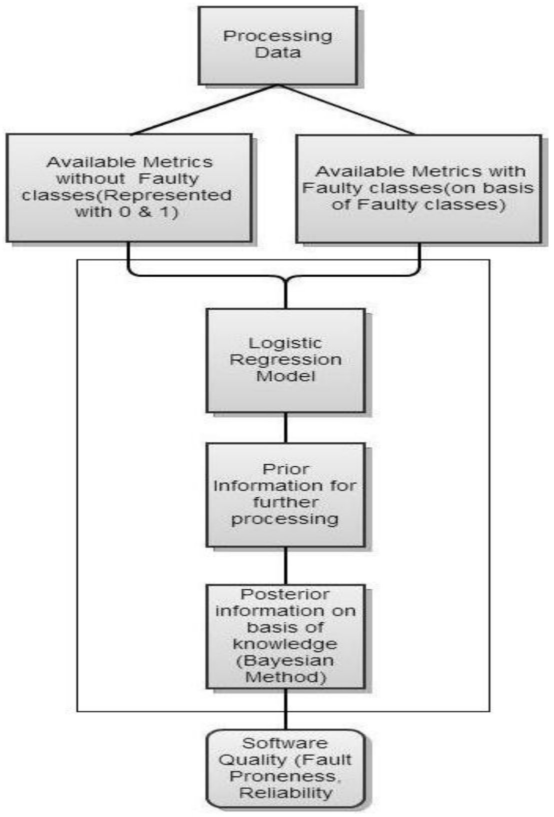
Processing Data
Available Metrics without Faulty classes(Represented with 0 & 1)
Available Metrics with Faulty classes(on basis of Faulty classes)
Logistic Regression Model
Prior Information for further processing
Posterior information on basis of knowledge (Bayesian Method)
Software Quality (Fault Proneness, Reliability

Fig. 1 Model Assessment Framework

The CK (Chidamber *et. al*. 1994) suite of metrics database was prepared with help of Analyst4j and computed each metric of its jdt core package. (Lincke *et. al*, 2008) compared the results of some selected software systems. Analyst4j can be used as a stand-alone Rich Client application or as an Eclipse IDE plug-in. Analyst4j is based on the Eclipse platform. Its features are find metrics, examine quality, and report creation for Java programs (Lincke *et. al*, 2008). Analyst4j measures these metrics, which forms the source for analysis (Kapila *et. al*, 2013). If there was at least one bad smell present in the code in a class, it was marked as smelly class. According to Moha (Moha *et. al*, 2010) “Code and design smells are poor solutions to recurring implementation and design problems.” In this study, the metric and bad smell database was collected by making use of the Analyst4j tool (Lincke *et. al*, 2008). A class is smelly if there is at least one type of bad smell. The binary categorization is used to classify classes into either error or non error category. The bad smells (kapila *et. al*, 2013) checked for include: blob classes, Spaghetti code, High risk function, complex classes, complex and undocumented code, Swiss knife classes.

4. Statistical Model

Two different types of methodologies were used to perform this experiment.

4.1 Software Tool Used

The Tool which is used to collect database for implementing this study is Analyst4j tool, which help us to collect all CK metrics values and find the bad smells present in software code of Eclipse 3.4. After collecting and preparing the database (shown in Fig 1.), evaluation was done with Matlab.

4.2 Model Based Upon Logistic Regression

Two different types of models are used to perform this experiment. The logistic regression model (Guido *et. al*, 2006) is used to study the relationship between bad smells and probability of software failure. It used a binary dependent variable, which represents whether classes are erroneous or not. Univariate binary logistic regression (UBR) (Singh *et. al* 2011), is useful for analyzing the data that includes binary variable.
The UBR model is as follows

$$\pi.\mathrm{x}=\frac{e^{g\,(x)}}{1+e^{g(x)}} \qquad (1)$$

Where g(x) = α + β * x a logit function and π represents probability of a class being faulty. X is an object oriented metric. Table 1 represents the common descriptive statistics of the investigated metrics. Firstly, univariate logistic regression is performed (see Table 1). The Coefficient is the predictable regression coefficient represented by α and β. Logistic regression technique were implemented with a threshold value of 0.5, which means that if $0.5 < \pi$, the class is classified as defective.

Table 1 Statistics for Eclipse 3.4

Test Metrics	Mean	SD	Max	Min	α	β
CBO	6.83	20.30	213	0	-3.7	0.183
WMC	36.76	122.95	1387	0	-1.8	0.015
RFC	38.07	125.58	1506	0	-2.0	0.023
NOC	0.16	0.949	9	0	-4.1	0.548
LCOM	0.40	0.451	1.5	0	-3.2	3.207
DIT	1.26	0.959	4	0	-3.4	-0.23

4.3 Model Based Upon Bayesian

In this study, a Bayesian inference model was designed to associate Object-Oriented software metrics to software fault content and fault proneness (Pai *et. al* 2007). Bayesian Inference represents a joint probability distribution over a set of variables, which are either distinct or continuous. A Bayesian inference model is a framework for constructing posterior data by combining prior knowledge with evidence. Bayes Theorem (Lee 2012) provides the basis for Bayesian inference model. According to Bayes theorem, the probability of an event is affected by evidence. The Bayesian approach offers intuitive and meaningful inferences about the data. Bayes Theorem is as follows:

$$P(X|Y) = \frac{P(Y|X)P(X)}{P(Y)} \qquad (2)$$

Where $P(Y|X)$ represents Likelihood, $P(X)$ represents Prior distribution and P(Y) is known as marginal likelihood or Prior predictive distribution.

5. Result & Conclusion

This study proposed the design for Bayesian inference[2] for individual metrics, which provide the posterior probability for smelly classes. Table 2 represents the Bayesian Inference graphs result with high prior and posterior values. This table includes all statistical information about each Bayesian Inference graph for all CK metrics. This table informs the prior and posterior range. Figures 2 to 6 represented the relation of prior, likelihood and posterior. The posterior is basically a combination of data and prior knowledge. The shape of the likelihood represents amount of information contained in the data. The amount of information is small, the shape of likelihood will largely dispersed, whereas if the amount of information is large, the likelihood function will closely focus around some particular value of the parameter. The representation of the Likelihood is flat in the Bayesian Inference graph[2] relative to the prior; it has little effect on the level of knowledge. In case the Prior and Likelihood has similar shapes the Posterior distribution is not greatly influenced by the Prior knowledge. In Bayesian Inference graphs, if Posterior and Likelihood have higher peak than prior, data is greatly influenced by prior data. The Posterior is a formal compromise between the Likelihood and the Prior. As shown in graphs, high point has the high posterior density.

In Bayesian Inference graphs, X-axis represents the probability of smelly classes and Y-axis represents density of fault occurrence. In Figure 2, 0 represented fault-free classes and as the graph moves towards 1 the probability of fault occurrence increases.

Figure 2 shows, Bayesian Inference graph for CBO. In this graph the high posterior density is designed for maximum number of classes with posterior probability value 0.39 and high prior density is designed for less number of classes with prior probability value 0.40. A High Posterior probability shows a large number of classes for which probability of failure is low. The graph for prior distribution is flat and represents less knowledge. The posterior distribution is relatively peaked, representing a boost in knowledge on the parameter and its situation is intermediate between the prior distribution and the likelihood of the data being an average of the two sources. The Bayesian Inference graph for CBO represents the similar shape for the likelihood and prior distribution. If the amount of knowledge obtained from data is small, then the posterior distribution will not change greatly from the prior distribution[2]. Prior and likelihood have different shapes so the posterior distribution is greatly influenced by the prior distribution.

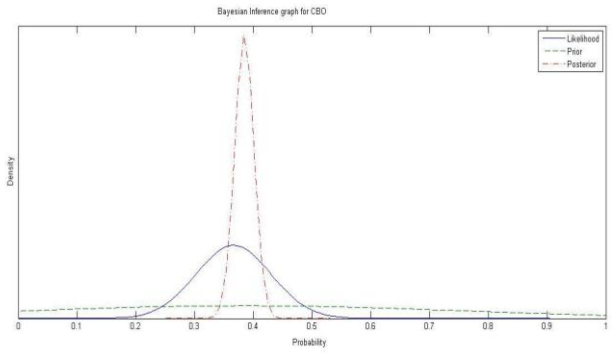

Fig 2. Bayesian Inference for representing Likelihood, Prior and Posterior for CBO metric of Eclipse

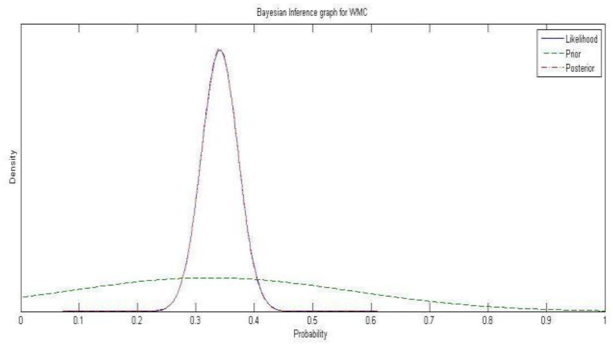

Fig. 3. Bayesian Inference for representing Likelihood, Prior and Posterior for WMC metric of Eclipse

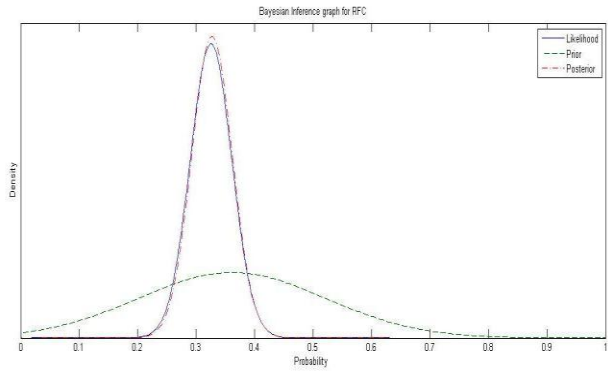

Fig. 4. Bayesian Inference for representing Likelihood, Prior and Posterior for RFC metric of Eclipse

Figure 3, Figure 4 and Figure 6 represent the Bayesian Inference graphs for WMC, RFC, and LCOM. The WMC, RFC and LCOM metrics have the same graph shape. The likelihood is highly peaked relative to the prior distribution. The posterior distribution is greatly influenced by the prior distribution on the level of knowledge. Bayesian Inference graphs for WMC, RFC and LCOM get have a posterior distribution that is narrower than the prior distribution. All the observations predict that future releases of the same software code will give us more reliable code and need less testing effort.

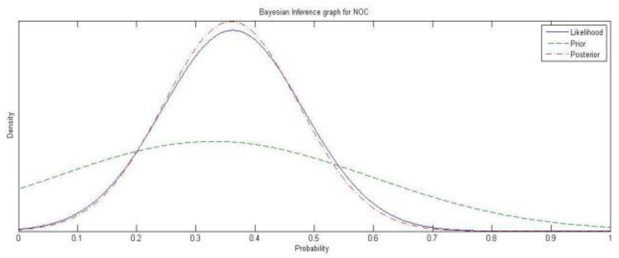

Fig. 5 Bayesian Inference for representing Likelihood, Prior and Posterior for NOC metric of Eclipse

Figure 5 shows Bayesian Inference graph represent for NOC. The graph for prior distribution represents its value below zero which means less knowledge obtained from data. The Posterior and likelihood distribution have similar shape. Figure 7 shows Bayesian Inference graph represent for DIT. The prior distribution is highly peaked relative to the likelihood and focused on 0. The closer the shape of the likelihood functions to the prior distribution, the smaller the amount of knowledge the data contains and so the posterior distribution will not change greatly from the prior. In conclusion, we presented a Bayesian Inference graphs that can be very informative to predict smelly classes. The approach is appropriate for applying data from ongoing projects

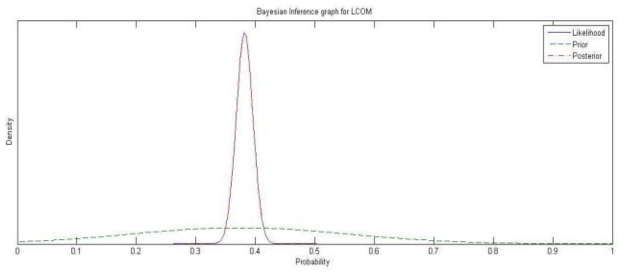

Fig. 6 Bayesian Inference for representing Likelihood, Prior and Posterior for LCOM metric of Eclipse

Table 2 Bayesian Inference graph result with Prior and Posterior values

Metric	High Posterior value	Posterior Range	High Prior value	Prior range
CBO	0.39	0.33-0.45	0.40	less than 0 to 1
WMC	0.31	0.20-0.42	0.32	less than 0 to 1
RFC	0.32	0.20-0.45	0.37	0 to 0.8
NOC	0.47	0.10 -0.90	0.34	less than 0 to 1
LCOM	0.37	0.31-0.42	0.41	less than 0 to 0.8
DIT	0.02	0.01-0.02	0.02	0 to 0.05

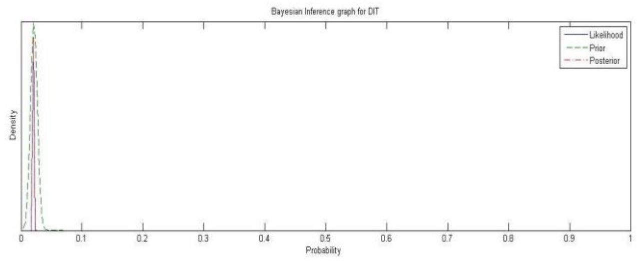

Fig. 7 Bayesian Inference for representing Likelihood, Prior and Posterior for DIT metric of Eclipse

References

Alshayeb, M., & Li, W. (2003). An empirical validation of object-oriented metrics in two different iterative software processes. *Software Engineering, IEEE Transactions on*, *29*(11), 1043-1049.

Basili, V. R., Briand, L. C., & Melo, W. L. (1996). A validation of object-oriented design metrics as quality indicators. *Software Engineering, IEEE Transactions on*, *2*(10), 751-761.

Bender, R. (1999). Quantitative risk assessment in epidemiological studies investigating threshold effects. Biometrical Journal, 41(3), 305-319.

Bennett, K. H., & Rajlich, V. T. (2000, May). Software maintenance and evolution: a roadmap. In *Proceedings of the Conference on the Future of Software Engineering* (pp. 73-87). ACM.

Catal, C., Sevim, U., & Diri, B. (2009). Software fault prediction of unlabeled program modules. In *Proceedings of the World Congress on Engineering* (Vol. 1, pp. 1-3).

Catal, C., & Diri, B. (2009). A systematic review of software fault prediction studies. *Expert systems with applications*, *36*(4), 7346-7354.

Chidamber, S. R., & Kemerer, C. F. (1994). A metrics suite for object oriented design. *Software Engineering, IEEE Transactions on*, *20*(6), 476-493.

Chidamber, S. R., Darcy, D. P., & Kemerer, C. F. (1998). Managerial use of metrics for object-oriented software: An exploratory analysis. *Software Engineering, IEEE Transactions on*, *24*(8), 629-639

Dejaeger, K., Verbraken, T., & Baesens, B. (2013). Toward Comprehensible Software Fault Prediction Models Using Bayesian Network Classifiers. *Software Engineering, IEEE Transactions on*, *39*(2), 237-257

Fenton, N. E., & Neil, M. (1999). A critique of software defect prediction models. *Software Engineering, IEEE Transactions on*, *25*(5), 675-689.

Guido, J. J., Winters, P. C., & Rains, A. B. (2006). Logistic Regression Basics. *MSc University of Rochester Medical Center, Rochester, NY*

Gyimothy, T., Ferenc, R., & Siket, I. (2005). Empirical validation of object-oriented metrics on open source software for fault prediction. *Software Engineering, IEEE Transactions on*, *31*(10), 897-910.

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

Kaur, S., & Kumar, D. (2011) Quality Prediction of Object Oriented Software Using Density Based Clustering Approach.

Lee, P. M. (2012). *Bayesian statistics: an introduction*. John Wiley & Sons

Lincke, R., Lundberg, J., & Löwe, W. (2008, July). Comparing software metrics tools. In *Proceedings of the 2008 international symposium on Software testing and analysis* (pp. 131-142). ACM.

Pai, G. J., & Bechta Dugan, J. (2007). Empirical analysis of software fault content and fault proneness using Bayesian methods. *Software Engineering, IEEE Transactions on*, *33*(10), 675-686.

Pandey, A. K., & Goyal, N. K. (2009). A fuzzy model for early software fault prediction using process maturity and software metrics. *International Journal of Electronics Engineering*, *1*(2), 239-245.

Rosenberg, L. H. (1998). Applying and interpreting object oriented metrics.

Shatnawi, R. (2010). A quantitative investigation of the acceptable risk levels of object-oriented metrics in open-source systems. *Software Engineering, IEEE Transactions on*, *36*(2), 216-225.

Shatnawi, R., Li, W., Swain, J., & Newman, T. (2010). Finding software metrics threshold values using ROC curves. *Journal of software maintenance and evolution: Research and practice*, *22*(1), 1-16.

Singh, S., & Kahlon, K. S. (2011). Effectiveness of encapsulation and object-oriented metrics to refactor code and identify error prone classes using bad smells. *ACM SIGSOFT Software Engineering Notes*, *36*(5), 1-10.

Subramanyam, R., & Krishnan, M. S. (2003). Empirical analysis of ck metrics for object-oriented design complexity: Implications for software defects. *Software Engineering, IEEE Transactions on*, *29*(4), 297-310.

Weber, P., Medina-Oliva, G., Simon, C., & Iung, B. (2012). Overview on Bayesian networks applications for dependability, risk analysis and maintenance areas. *Engineering Applications of Artificial Intelligence*, *25*(4), 671-682.

Moha, N., Gueheneuc, Y. G., Duchien, L., & Le Meur, A. (2010). DECOR: A method for the specification and detection of code and design smells. *Software Engineering, IEEE Transactions*,*36*(1), 20-36