ISSN 2278-3091
Volume 9, No.4, July – August 2020
International Journal of Advanced Trends in Computer Science and Engineering
Available Online at http://www.warse.org/IJATCSE/static/pdf/file/ijatcse34942020.pdf
https://doi.org/10.30534/ijatcse/2020/34942020

Prediction of Source Code Quality Using Cyclomatic Complexity and Machine Learning

G.Vamsi Krishna, M.Chaitanya Meher, Khushboo Jain, D.Nagamallesawari
Department of Computer Science and Engineering,
KoneruLakshmaiah Education Foundation,
Guntur, Vaddeswaram, A.P., India

ABSTRACT

The automated analysis of source code quality can help the developers to decrease the potential source code anomalies without the help of a peer reviewer. This paper analyses current tools that are being used for source code analysis and also proposes a method to use classification and cyclomatic complexity to predict the quality of the source code by giving public data sets as input. This automated prediction of source code quality will help the developer in providing feedback during the development life cycle without the help of a peer reviewer.

Key words: Source Code Analysis Cyclomatic Complexity, Classification.

1. INTRODUCTION

Predicting the quality of source code was a fascinating topic for many years, a wide range of algorithms and solutions are developed in this field. But many of those solutions' objective is to only focus on fault prediction in the source code. Prediction of source code quality before performing testing can reduce the cost and time required to test the application. Quality of the source code depends on many factors including complexity, functional requirements, and structure of the code. Prediction of source code quality in the early stages of the development life cycle can help the developers to reduce the potential source code anomalies which may generate field failures. [7] Source code quality must be checked and reviewed as it is one of the important mechanisms for quality assurance in the software development life cycle. Reviewing the quality of source code has the potentiality to provide feedback to the developer and helps the developer to avoid introducing new bugs. The underlying problem in reviewing the quality of the source code is that not all reviewers can understand the minute changes that can affect the functionality of the application. Throughout the sense of software engineering, the quality of the software may be specified as well as how well the software fulfills the user needs and the specifications on which it is built.[6] The most powerful indicators of the quality of the source code is the complexity of the code which can be found by using cyclomatic complexity. Cyclomatic Complexity is the software utilized and is developed by Thomas J, McCabe Sir in the year 1976 by utilizing the complication of the program. It will be a quantitative measure of the multifaceted nature of modifying guidelines. It specifically measures that number of linearly autonomous ways through a program's source code. Cyclomatic Complexity is a product metric which gives quantitative measurements of a system's legitimate multifaceted nature. This research was focused on learning both programming languages and written languages in their complexities. A code smell is an indication that software can contain a potential problem. Software smells typically aren't glitches since they are technologically right and don't hinder the system from running. Rather they show a deficiency in the architecture or a growing possibility of potential failures. Bloasters in the source code must be handled before deployment to increase the performance of the system.[1]. The ability of the program usually degenerates as the program undergoes improvements over its lifespan. Successfully managing such a program means that current code, in addition to incorporating new features, needs to be constantly refactored, i.e. enhanced without incorporating features. They classified related bad smells and carried out an observational analysis that established an initial connection between the code smells.

2. RELATED WORK

Previous studies have examined the possible solutions to predict the quality and reliability of the source code. The agile analysis explores the code of the software and explanations behind any potential problems that could arise during the runtime. Defaults in applications can be detected using methods focused on static analysis. Recent advancements in technologies have put in devices to do deeper analyzes and uncover more flaws and create a small number of false alarms meaning only that are absolutely accurate and needed. The purpose of Ivo Gomes et al. work is to explain briefly static code analysis, its features and possibilities, giving an

overview of the principles and technologies behind this method of software development approach, as well as the methods that enable the use of code review tools to help programmers create applications which provide the developers a platform to improve the source code and correct errors during the development life cycle[2]

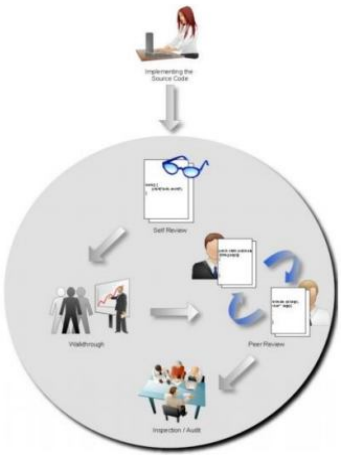

Figure 1: Traditional Peer Review

The list of components in a traditional peer review report shown in Figure 1 contains: checklist which should test and reflect on all products, list in defects detected, list of participants, examination indicators, state of the item being examined. Alberto et al.[3] After conducting extensive experiments reported on CodeFlow that reviewing the quality of code at earlier stages in the development life cycle has the potentiality to provide feedback to the developer and helps the developer to avoid introducing new bugs. CodeFlow is a shared code review tool that enables users to annotate source code directly in their viewer and to engage in a live chat environment with the analysis participants. CodeFlow stores all the code feedback details on a database repository. This provides an additional data source that we used to analyze commentary on real code review without the Hawthorne effect.

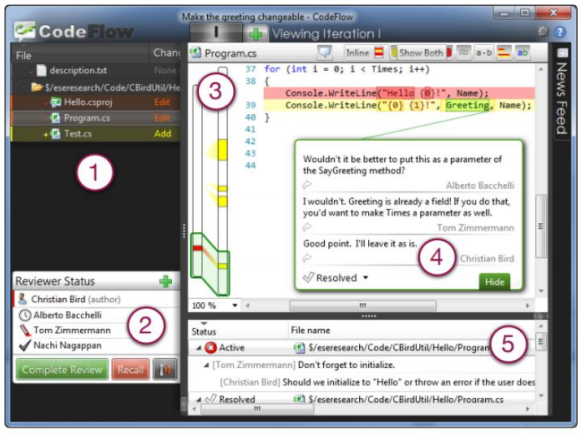

Figure 2: CodeFlow automated testing tool

CodeFlow is used by developers at Microsoft as the main review to tool to find the quality of the source code in the early stages of the development life cycle.

3. OUR APPROACH

Our approach is to use cyclomatic complexity and classification in machine learning to predict the quality and reliability of the source code whether it well written or badly written. Prediction of source code quality before performing testing can reduce the cost and time required to test the application. Quality of the source code depends on many factors including complexity, functional requirements, and structure of the code.

3.1. Classification

In machine learning, classification is the process of defining which class belongs to a new and unlabeled object, based on prior knowledge. Methods used for text classification and static analysis include Bayesian classification, n-gram, Vector Machine Help (SVM) and Decision Tree.[4]

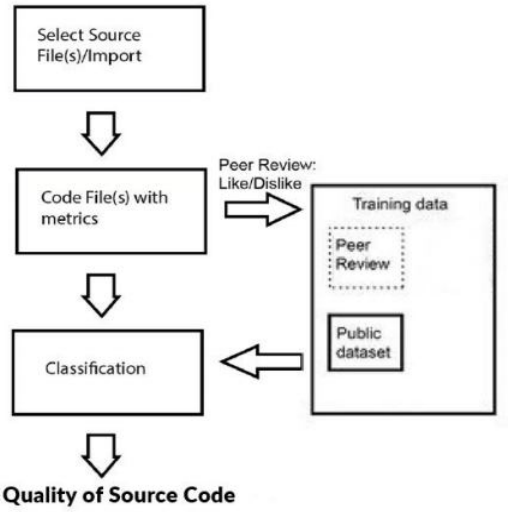

Figure 3: Classification of Source Code

Classification is the method of defining the category to which a new instance belongs, utilizing only the training data given. The methods can be described by using the metrics produced from the static analysis and the training data available as shown in Figure 3.

Table 1: Example of Classification of Public Data Set Promise

	Prediction	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11
Method 1	Well written	1	11	316	0.03	36	0.06	9	30	17.6	3	0
Method 2	Badly written	2	8	199	0.11	4	1.19	5	87	11.1	1	1
Method 3	Well written	7	42	1021	0.03	88	0.05	18	40	56.7	0	11
Method 4	Well written	1	10	298	0.03	78	0.03	6	0	16.6	4	0
Method 5	Well written	1	8	230	0.04	70	0.03	4	24	12.8	6	0

In the Table 1, prediction of the source code quality is obtained after combining selected features from C1,C2,C3,C4,C5,C6,C7,C8,C9,C10,C11.But there are more, we have six more features adding to the list C12,C13,C14,C15.

Table 2: Selected Features and their description

Feature	Description
C1	McCabe Cyclomatic complexity
C2	Halstead total operators + operands
C3	Halstead Volume
C4	Halstead program length
C5	Halstead difficulty
C6	Halstead intelligence
C7	Halstead effort
C8	Halstead bug
C9	Halstead time estimator
C10	Halstead line cont
C11	Halstead lines of comments
C12	Halstead lines of blank lines
C13	Unique operators and operands
C14	Total operators
C15	Total operands

3.2. Cyclomatic Complexity

Cyclomatic Complexity is a metric utilized and developed by Thomas J, McCabe Sir in the year 1976 by utilizing the complication of the program. It will be a quantitative measure of the multifaceted nature of modifying guidelines. It specifically measures that number of linearly autonomous ways through a program's source code. Cyclomatic Complexity is a product metric which gives quantitative measurements of a system's legitimate multifaceted nature. Cyclomatic Complexity Number M for any source program can be calculated theoretically using the formula,

M=E-N+2P

E- edges generated in the control graph

N- nodes generated in the control graph

P- connected components

Step 1: A graph has to be constructed from the skeleton source program
Step 2: Independent Paths, Nodes, and edges are to be identified
Step 3: From the number of independent paths the cyclomatic complexity can be calculated.

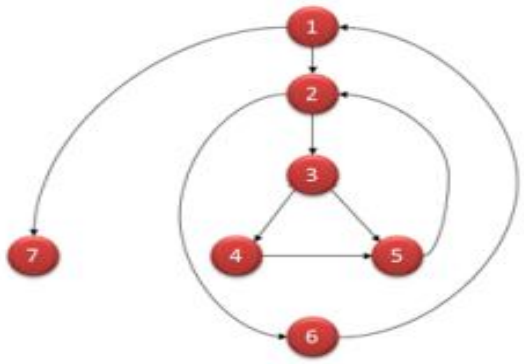

Figure 4: Example of Flow Graph

The Properties of Cyclomatic complexity are listed below: V (G) is the maximum number of independent paths in the graph
V (G) must be greater than or equal to one
The graph will have only one path if the formula
V (G)=E-N+2P generates 1
The cyclomatic complexity number must be minimized 10 for optimal results

Table 3: Complexity Number and its description

Complexity Number	What does it mean?
1-10	Testing: very easy ost required to test: very low Effort estimation: very low
10-20	Testing:easy Cost required to test: low Effort estimation: low
20-40	Testing:difficuly Cost required to test: high Effort estimation: high
>40	Testing:difficult Cost required to test: high Effort estimation: very high

3.3. Architectural overview

In order to even further examine the prediction capacity of approach, we concentrated on using both cyclomatic complexity and the classification methods. JHawk produces the static analysis for the source code, and the module produces the required metrics C1,C2,C3,… ,C14,C15. The Architectural overview is shown in Figure 5

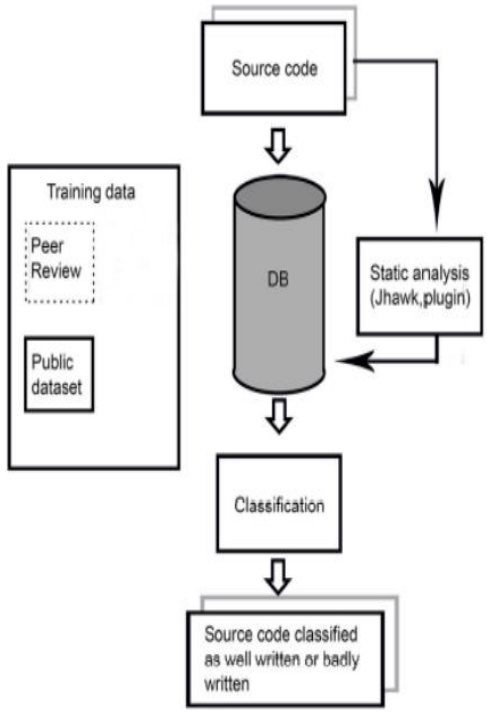

Figure 5: Architectural overview

The source code of which the quality is to be predicted is given as input to the database along with the training data sets using jhawk plugins classification is done and the quality of the source code is generated.

4. EMPIRICAL STUDY

In this section, we conduct an empirical study to answer the research questions that are listed below. All the experiments are performed on a Custom Desktop Computer with AMD Ryzen Threadripper 3960X Processor and Adata XPG Gammix D30 16GB (16GBX1) DDR4 RAM clocked at 3200MHz and Lenovo Laptop with Intel Core i7-8565U processor overclocked to 4.60 GHz speed processor and a Samsung Single Channel 8GB DDR4 RAM at 2133MHz.

Tools: MySql Database
Classification Methods: K nearest neighbor (KNN), Naive Bayes (NB), and Decision tree (DTree).
Programming Language: Python
Data Sets: PROMISE1: which is public dataset available in PROMISE workshop website
Link: https://datahub.io/machine-learning/jm1
PROMISE2: Public dataset jm1 dataset Link:https://datahub.io/machine-learning/jm1

Q1) Is it possible to predict the source code quality using classification methods?

Pre=Precision, Rec=Recall

Table 4: Prediction of Source code after classification

Dataset	Features	Method	Badly written Pre	Rec	Well written Pre	Rec	Total
PROMISE1	21	KNN	**0.228**	0.356	0.906	0.837	77.95 %
		DTree	**0.364**	0.366	0.856	0.855	76.40 %
		NB	**0.084**	0.402	0.971	0.822	80.61 %
PROMISE1	15	KNN	**0.227**	0.348	0.903	0.836	77.68 %
		DTree	**0.374**	0.362	0.849	0.856	76.06 %
		NB	**0.087**	0.415	0.972	0.823	80.72 %
PROMISE1	9	KNN	**0.287**	0.423	0.910	0.848	79.43 %
		DTree	**0.339**	0.336	0.847	0.848	75.24 %
		NB	**0.323**	0.361	0.869	0.849	76.76 %
PROMISE2	9	KNN	**0.274**	0.409	0.909	0.846	79.11 %
		DTree	**0.364**	0.350	0.845	0.853	75.54 %
		NB	**0.506**	0.323	0.757	0.870	71.02 %
PROMISE2	15	KNN	**0.292**	0.419	0.907	0.848	79.28 %
		DTree	**0.361**	0.355	0.850	0.853	75.90 %
		NB	**0.523**	0.315	0.739	0.871	69.90 %

The classification of the public data sets PROMISE1 and PROMISE2 was done by using KNN(k nearest neighbor), Decision Tree, and NB (naïve Bayes). When we take a look at the prediction Table 4 it might look like Naïve Bayes has the best results with 80.72 % on the dataset with value by using 15 features. If we perform an in-depth analysis on Pre(Precision) and Rec(Recall) the result might not seem like naïve Bayes performance is the best out of all three including KNN and Decision Tree. When using the PROMISE1 data set the KNN classification potential was 22.7% and 90.3% for badly written methods and well-written methods respectively. The Decision Tree classification on the PROMISE1 data set was able to classify 37.4% of the badly written methods with good precision and 84.9% of the well-written methods with good precision. When using PROMISE1 data set the naïve classification was able to classify 8.7% of the badly written methods with good precision and 97.2% of the well-written methods with good precision. For the PROMISE2 dataset using naïve Bayes classification, there was a trade-off between the precision for badly written methods and well-written methods. In Table 4 we can clearly see that the Pre for the badly written method was increased whereas the Pre of the well-written methods were decreased, that kind of trade-off in performance needs analyzing. When using PROMISE2 data set the naïve Bayes classification was able to predict with an all time low,52.3% of the badly written methods. On the other hand classification rate of well-written methods with good precision is about 73.3% which is a huge drop off when compared to PROMISE1 dataset.

5. CONCLUSION

Machine learning and cyclomatic complexity have proved to predict the quality of the source code precisely. In this paper, we proposed an approach to predict the quality of the source code using machine learning and cyclomatic complexity using public data sets PROMISE1 and PROMISE2.KNN,

decision tree, and naïve Bayes were the classification methods used on the public data sets, naïve Bayes was able to perform better than KNN and decision tree. The public data sets contained more well-written methods than badly written methods so the classifiers identified well-written code when compared to badly written code. We conclude that predicting the source code quality accurately using classification is possible.

REFERENCES

1) Mantyla, Mika, Jari Vanhanen, and Casper Lassenius. "**A taxonomy and an initial empirical study of bad smells in code.**" International Conference on Software Maintenance, 2003. ICSM 2003. Proceedings. IEEE, 2003.
2) Gomes, I., Morgado, P., Gomes, T., & Moreira, R. (2009). **An overview on the static code analysis approach in software development.** Faculdade de Engenharia da Universidade do Porto, Portugal.
3) Bacchelli, Alberto, and Christian Bird. **"Expectations, outcomes, and challenges of modern code review."** 2013 35th International Conference on Software Engineering (ICSE). IEEE, 2013. https://doi.org/10.1109/ICSE.2013.6606617
4) Barstad, Vera, Morten Goodwin, and Terje Gjøsæter. **"Predicting Source Code Quality with Static Analysis and Machine Learning."** NIK. 2014.(should keep somewhere)
5) Graylin, J. A. Y., et al. **"Cyclomatic complexity and lines of code: empirical evidence of a stable linear relationship."** Journal of Software Engineering and Applications 2.03 (2009): 137.(should keep near cyclomatic complexity) https://doi.org/10.4236/jsea.2009.23020
6) Chaitanya Krishna, B., Yeshwanth Srinath, A., Bhavani, N., & Jaya Sai, G. (2018), **"Analysing software quality using CMMI-2 with agile-scrum framework".** International Journal of Engineering and Technology(UAE), 7(1.1 Special Issue 1), 290-293. https://doi.org/10.14419/ijet.v7i1.1.9704
7) D. Binkley, Source code analysis: **A road map, "Future of Software Engineering FOSE"**, vol. 7, pp. 104{119, 2007. https://doi.org/10.1109/FOSE.2007.27
8) **"Design of data acquisition process and its validation through statistical approaches"**, Naga Malleswari, D., Subrahmanyam, K. ,International Journal of Recent Technology and Engineering (2019)
9) **"Analysis of risk in information system using cyclomatic complexity "**,Naga Malleswari, D., Bhaskar, K., Monica, A., Venkat Vinay, B., Sai Anirud Varma, U. (2019), International Journal of Recent Technology and Engineering
10) **"SIS framework for risk assessment through quantitative analysis",** NagaMalleswari, D., Subrahmanyam, K. ,(2019) International Journal of Engineering and Technology(UAE)
11) **"Validation of SIS framework using ASP/JSP based information system",** NagaMalleswari, D., Subrahmanyam, K. (2019) International Journal of Innovative Technology and Exploring Engineering
12) "A Security Approach for File Management System using Data Encryption Standard (DES) algorithm", Irma T. Plata Edward B. Panganiban; Bryan B. Bartolome (2019) International Journal of Advanced Trends in Computer Science and Engineering
13) "A Fuzzy Analytic Hierarchy Process for Security Risk Assessment of Web based Hospital Management System", Intisar Shadeed Al-Mejibli; Nawaf Rasheed Alharbe (2019) International Journal of Advanced Trends in Computer Science and Engineering
14) "Information Security Using Hilbert With Hash Value",Koduru Prasada Rao; Dr G Lavanya Devi; (2019) International Journal of Advanced Trends in Computer Science and Engineering
15) "A Comprehensive Survey on Vertical Handover Security Attacks during Execution Phase",Omar Khattab (2019) International Journal of Advanced Trends in Computer Science and Engineering