THINK INDIA JOURNAL

ISSN:0971-1260
Vol-22- Issue-14-December-2019

Analyzing Maintainability and Compliance of Different Version of Software in Code Cloning

Dr.Sumesh Sood
IKGPTU Dinanagar Campus, Punjab, India

Sandeep Bal
Research Scholar, IKG-PTU, Kapurthala, Punjab, India

Abstract

Software code cloning is to write the code once and use it wherever required later on by making certain changes as per the requirement of the new/updated module/version. Such practices result in saving time, effort, resources and money, but the inconsistencies may bring in 'bugs or vulnerabilities' in the system which may further lead to decay (degradation) in software architecture. Performing the maintainability index calculations and compliance calculation provides information about the actual architecture decay during software code cloning. In this paper, two different software tools are used. 'Analyze' is used for calculating the Maintainability Index and 'CppDepend' is to find the Compliance value. To find the relationship between the Maintainability Index and Compliance calculations in software code cloning a case study of software 'DECEIVER' written in C++ is performed. Results obtained show that the inconsistent value of compliance, resulted in architecture decay of software.

Keywords: Cloning, Maintainability, Compliance.

1. Introduction

A software evolution takes place throughout its life cycle for a variety of reasons. Changes are made in the software code by performing additions and deletions in the source code files. As the version of a software change, count of total line of code (LOC) also changes though not mandatorily. But the effect of code editing/changes into various versions of software may lead to inconsistent code with increased complexity and few other problem areas. One such field is software code cloning. The idea behind cloning is to write the code once and use it wherever required later on by making certain changes as per the requirement of the new/updated module/version. It is like using software code again and again without re-writing from scratch at the time of developing a new version or developing a component of similar nature. Such practices result in saving time, effort, resources and money, but the inconsistencies may bring in 'bugs or vulnerabilities' in the system which may further lead to decay (degradation) in software architecture. This issue can be dealt with by exploring and comparing the software code of different versions of the software system and finding out the similarity percentage in the code i.e. generating metrics to evaluate a comparative study of the selected set of values. Performing the maintainability index calculations and compliance calculation will provide information about the actual architecture decay. After such an evaluation, the clones can be further managed by using/designing refactoring/reengineering techniques.

ISSN:0971-1260
Vol-22- Issue-14-December-2019

2. Literature Review

Mayrand et al. in 1996 presented a technique for 'automatic detection of duplicate and near-duplicate functions' in a software system. 'Datrix™' tool was used to initiate the detection technique where 21 function metrics were extracted from the source code. These metrics were grouped into four points of comparison i.e. 'name, layout, expression, and control flow'. Clones were categorized into different cloning levels ranging from an 'exact copy' to 'distinct' functions.

In 1997, Lague et al. discussed the evolution of function-level code clones. This study revealed that the growth of clones can be controlled by 'preventive support'. The discovery and correction of clones during the initial phases could be dealt with the concept of 'problem mining'.

To identify duplicate code Ducasse et al. developed a ' language-independent ' technique in 1999. The method is based on simple line-based string matching, visualization of the duplicated code and detailed textual reports.

Krinke analyzed a large number of revisions of five open-source systems in 2007and found more frequency of inconsistent changes to clone groups. In another study in 2008 on clone evolution, he found cloned code to be better and stable instead of non-cloned code.

Rahman et al. discussed the relationship between clone and defect in 2012. They noticed little or no relation between clones and bugs. In addition, their study also showed that cloned codes are less buggy and larger clone classes do not necessarily contribute more to defects.

Abd-El-Hafiz in 2012, proposed a 'metric-based technique' for clone detection. This technique dealt with different function metrics where a data mining fractal clustering algorithm was used to divide the source code into a small number of clusters. In a metric space, a cluster comprises of those functions which are within a specific distance from each other. The last phase is the extraction of clone classes from the clusters.

In 2014, Kaur and Kaur designed a tool in Java to detect clones in Web applications based on PHP and JHP as well as in Java programming language. It detected the higher-level clone called 'Directory Clones' in Java. This proposal is also a hybrid combination of metrics-based approach and textual comparison.

In 2015, Kaur and Lal also proposed a 'hybrid technique' which segments the code into a number of functions and finds code clone from the source code. In this technique, the metric comparison of the two source code was done for detecting potential code clones in the source code. The next step in the process was template conversion where the parser converted all constant values to word constant. The textual comparison was done after the template conversion which confirms the existence of code clone when the number of line matched crosses the threshold value or percentage value.

In 2016, Sushma and Bhagwan proposed a 'metrics-based technique' for file/ module level clone detection and developed a tool in Java named JSCCD (JS Code Clone Detector) which could detect clones between Java files.

Kodhai and Kanmani in 2017 proposed a tool named 'CloneManager' which used another 'hybrid approach combining a metric-based approach with textual analysis' performed for detecting both syntactical and functional clones in source code.The tool combined textual analysis with metrics to detect all the four types of clones by differentiating function clones from a given Java source code project.

ISSN:0971-1260
Vol-22- Issue-14-December-2019

In 2017 Kaur et al. suggested a 'metric-based technique' in combinationwith 'swarm and artificial intelligence techniques'. The metric-based approach extracted the code metrics with the help of code clone techniques based on PDG and AST tree.

Kaur and Sharma in 2018 proposed an approach that adopts 'Optimized SVM algorithm' where an optimized code manager based on support vector machine is used. The proposed method highlighted clone detection by combining metrics computation and textual analysis.

The recent trends in code clone detection and evaluation field shows that different techniques have been implemented in order to find clones with ease. This paper is focused on Maintainability Index and Compliance calculations to support clone detection and provide reliable results.

3. Case Study

To find the relationship between Maintainability Index and Compliance calculations in software code cloning a case study of software 'DECEIVER' written in C++ is performed. The code of the software is available in 'GitHub'. The reason for selecting this software is that it is open-source software and availability of using open source software is more as compare to proprietary software. This is because open source software and its source code are available free. So making changes in the software and creating new versions of the software is easier.

A feature 'Analyze' from Visual Studio 2015 is used to find the Maintainability Index of 'DECEIVER'. It gives a measure about software that how easily source code can be maintained. The formula for calculating Maintainability Index (Oman and Hagemeister, 1994) is:

MI = MAX (0, (171 – 5.2 * ln(Halstead Volume) – 0.23 * (Cyclomatic Complexity) – 16.2 * ln (Lines of Code)) * 100 / 171)

CppDepend tool provides the Compliance values. Compliance is a measure of software, which determines whether the software has been developed in accordance with the standard rules or not. It is measured on the basis of CppDepend's summary of violations of rules (Bhakthavatsalam et al., 2015; Smacchia, 2016). The following formula provides the Compliance of software in terms of percentage (Vattumalli, 2010):

Compliance = ((Total number of rules – Number of rules violated) / Total number of rules) * 100

4. Result and Analysis

CppDepend Tool is used in different versions of software DECEIVER. Following snapshots show the summary of rules for different versions of DECEIVER.

ISSN:0971-1260
Vol-22- Issue-14-December-2019

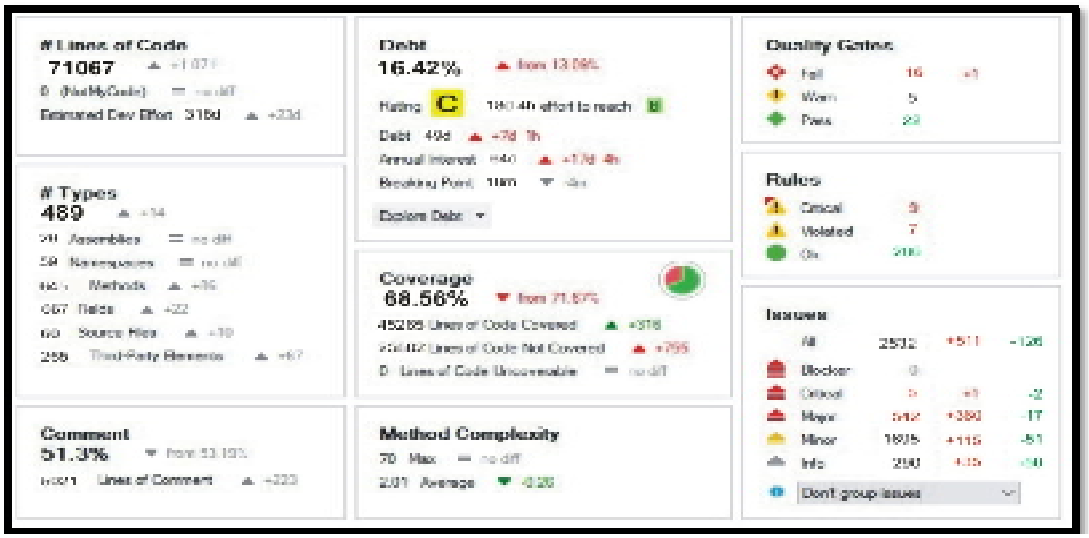

Figure 1: Summary o f Rules for Version Demo

Figure 1 contains the Summary of Rules for Version Demo which shows that 7 Rules are violated out of total 293 Rules while 286 Rules are Ok. The Compliance calculated is 97.6 %.

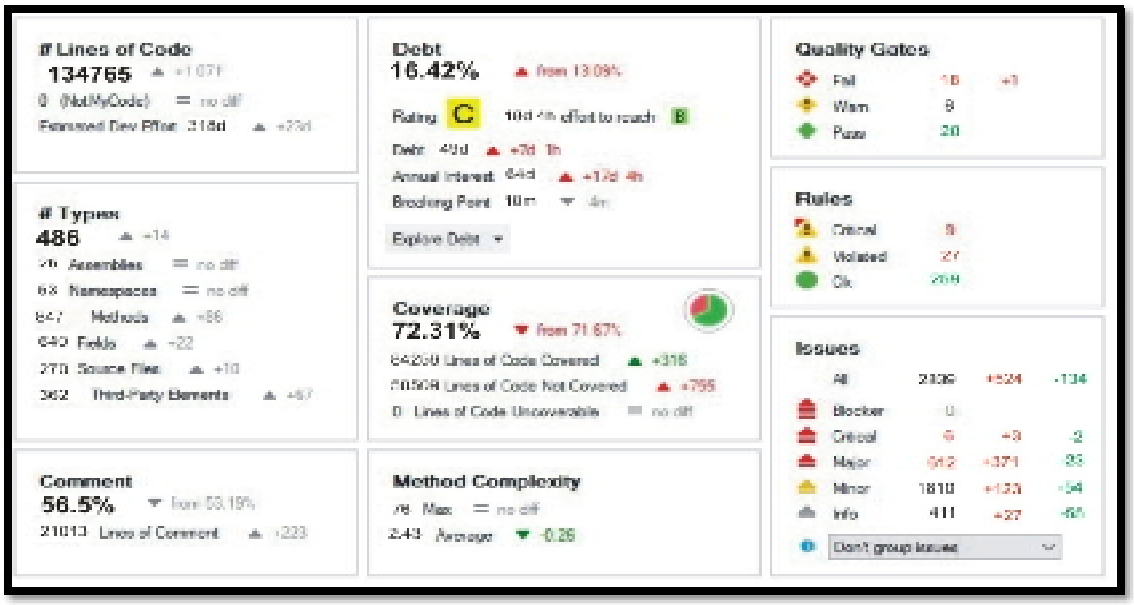

Figure 2: Summary o f Rules for Version 16_07_03

Figure 2 contains the Summary of Rules for Version 16_07_03 which shows that 27 Rules are violated out of 293 Rules while 259 Rules are Ok. The Compliance calculated is reduced with great margin to 90.7 %.

ISSN:0971-1260
Vol-22- Issue-14-December-2019

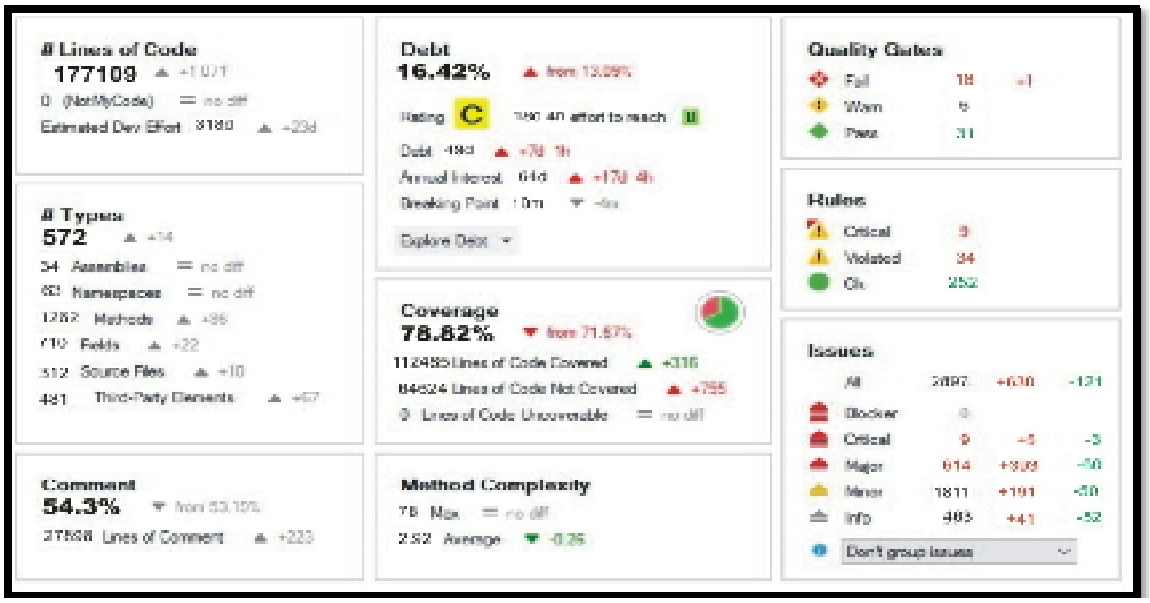

Figure 3: Summary of Rules for Version Alpha

Figure 3 contains the Summary of Rules for Version Alpha which shows that 34 Rules are violated out of total 293 Rules while 252 Rules are Ok. The Compliance calculated is reduced further to 88.3 % which badly affects the overall software architecture.

Figure 4 shows a graph depicting the CC and MI of all the versions of this case study. As the version change, its Cyclomatic Complexity rises marginally. This increased complexity, poorly affects the overall software architecture when the associated MI shows declining values.

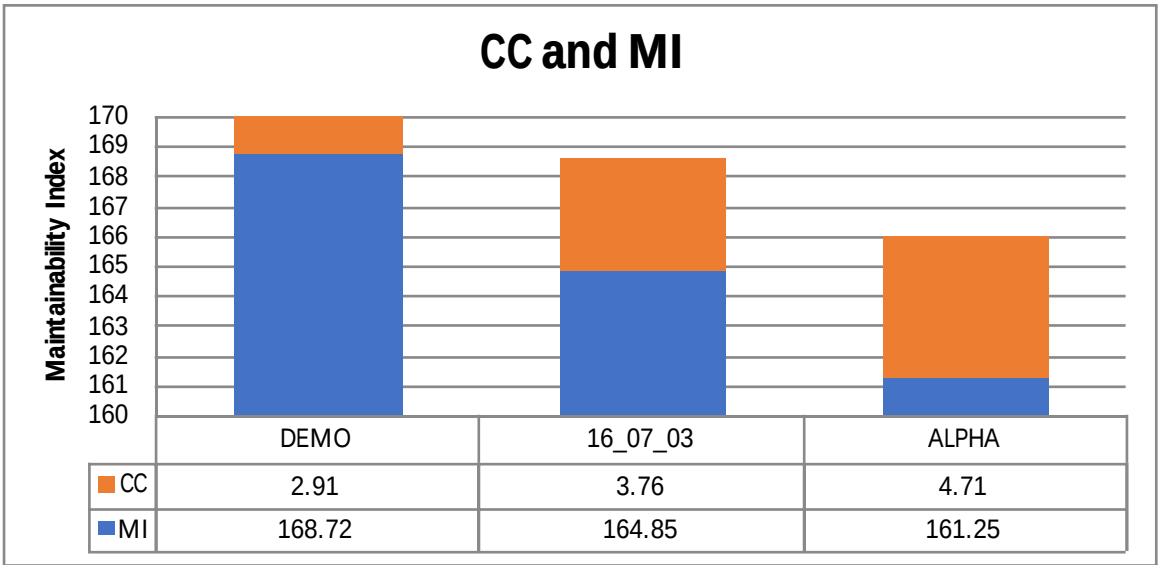

Figure 4: CC and MI for DECEIVER Versions

Table 1 shows the calculated Maintainability Index and the corresponding Compliance percentage for software.

ISSN:0971-1260
Vol-22- Issue-14-December-2019

Table 1: Comparison of Maintainability Index and Compliance of Different Version of DECEIVER

Sr. No.	Version Name	MI	CC	Size	Compliance
V1	DEMO	168.72	2.91	71067 LOC	97.6 %
V2	16_07_03	164.85	3.76	134765 LOC	90.7 %
V3	ALPHA	161.25	4.71	177109 LOC	88.3 %

E.g. the MI for all versions varied from 168.72 to 161.25 showing an overall decrease of 7%. The complexity also increased with each successive version proportionately to the increased size of software in each version. This has been the result due to the fallen values of compliance from 97.6 % to 88.3 %. Therefore, it is visible that the inconsistent value of compliance resulted in the architectural decay of software. Software cloning has affected the overall software architecture in this case where clone management is required for dealing with software maintenance activities.

Hence, this case study gives an example where Code Cloning (detected by the Metrics-based approach) across the software system has affected its architecture because of the code not written in conjunction with the software compliance factor.

5. Conclusion

Two metrics Maintainability Index and Compliance has been calculated to study the software maintenance status. The Maintenance Index is calculated using Visual Studio 2015's 'Analyze' feature. Another tool 'CppDepend' is used for C++ code to generate the Size, Cyclomatic Complexity and Compliance for each of the versions of 'DECEIVER' software and explore the coding architecture inside out. The obtained resultshows that the inconsistent value of compliance resulted in the architectural decay of software. This validated that the maintainability index calculations and compliance calculation will provide information about the actual architecture decay during software code cloning.

References

- Mayrand, J., Leblanc, C., and Merlo, E., (1996), Experiment on the Automatic Detection of Function Clones in a Software System Using Metrics. In: Proceedings of the International Conference on Software Maintenance (ICSM), pp. 244–253. IEEE CS, Monterey, CA, USA.
- Lague, B, Proulx, D.,Mayrand, J., Merlo, E. and Hudepohl, J. P., (1997), Assessing the benefits of incorporating function clone detection in a development process. In ICSM, pp. 314–321.
- Ducasse, S., Rieger, M., Demeyer, S. (1999), A Language Independent Approach for Detecting Duplicated Code. In: Proceedings of the 15th International Conference on Software Maintenance (ICSM), pp. 109–118. IEEE CS, Oxford, England, UK.
- Krinke, J. (2007), A study of consistent and inconsistent changes to code clones, 14th Working Conference on Reverse Engineering (WCRE 2007), pp. 170-178, IEEE.
- Krinke, J. (2008), Is cloned code more stable than non-cloned code, Eighth International Working Conference on Source Code Analysis and Manipulation. pp. 57-66, IEEE.

ISSN:0971-1260
Vol-22- Issue-14-December-2019

- Rahman, F., Bird, C., and Devanbu, P. (2012), Clones: What is that smell, Empirical Software Engineering, pp. 503-530.
- Abd-El-Hafiz, S. K. (2012), A metrics-based data mining approach for software clone detection, 36th Annual Computer Software and Applications Conference, pp. 35-41, IEEE.
- Kaur, K., and Kaur, R. (2014), Clone detection in web application using clone metrics. International Journal of Advanced Research in Computer Science and Software Engineering, pp. 974-982.
- Kaur, M., and Lal, M. (2015), Code clone detection using function based similarities and metrics. International Journal of Emerging Research in Management and Technology, pp. 156-159.
- Sushma, and Bhagwan, J. (2016),A novel metrics based technique for code clone detection. International Journal of Engineering and Computer Science, pp. 18221-18224.
- Kodhai, E., and Kanmani, S. (2017), Method-level code clone detection for java through hybrid approach. International Arab Journal of Information Technology. (pp. 914-922).
- Kaur, G. and Sharma, S. (2018). Metric level based code clone detection using optimized code manager. International Journal of Engineering and Technology, pp. 144-149.
- Oman P. and Hagemeister J., (1994), Constructing and Testing of Polynomials Predicting Software Maintainability, Journal of Systems and Software, 24(3), pp. 251-266.
- Bhakthavatsalam, N., Jayaraaman, A., D'Souza, G., Raghavachar, P., Gopal, P., Gosselin, J., Garrison, J., and Cloutier, L., (2015). Managing compliance: In oracle enterprise manager lifecycle management administrator's guide, from web site https://docs.oracle.com/cd/E24628_01/em.121/e27046/compliance_lcm.htm#EMLCM93381, accessed on 20/3/2019.
- Smacchia, P., (2016), NDepend (Version 6.3), Computer Software, from web sitehttp://www.ndepend.com/download, accessed on 28/06/2019.
- Vattumalli, N.B., (2010), Panorama - A software maintenance tool, Master's thesis, Iowa State University, Iowa, USA.

