Complexity Metrics Significance for Defects: An Empirical View

Chapter 1
Complexity metrics significance for defects: An empirical view

Syed Muhammad Ali Shah[1] and Maurizio Morisio

Abstract. Software Complexity often seems to be correlated with the defects and this makes difficult to select appropriate complexity metrics that would be effective indicators of defects. The aim of this work is to analyze the relationship of different complexity metrics with the defects for three categories of software projects i.e. large, medium and small. We analyzed 18 complexity metrics and defects from 27,734 software modules of 38 software projects categorized in large, medium and small. In all categories of projects we do not find any strong positive correlation between complexity metrics and defects. However we cluster the complexity metric values and defects in three categories as high, medium and low. Consequently we observe that for some complexity metrics high complexity results in higher defects. We called these metrics as effective indicators of defects. In the small category of projects we found LCOM as effective indicator, in the medium category of project we found WMC, CBO, RFC, CA, CE, NPM, DAM, MOA, IC, Avg CC as effective indicators of defects and for a large category of projects we found WMC, CBO, RFC, CA, NPM, AMC, Avg CC as effective indicators of defects. The difference shows that complexity metrics relation to defects also varies with the size of projects.

Keywords: Complexity • defects • metrics

[1] S. M. A. Shah(✉)
Politecnico di Torino, Corso Duca degli Abruzzi, 24 10129 Torino, Italy
e-mail: syed.shah@polito.it

M. Morisio
Politecnico di Torino, Corso Duca degli Abruzzi, 24 10129 Torino, Italy
e-mail: maurizio.morisio@polito.it

1. INTRODUCTION

For every software product, the quality is critically important. Quality can be characterized by different measures related to defects. However, it is hard to measure defects ahead of time (1). Therefore, many internal properties are used to predict quality e.g. size and complexity.

Size mostly presented in LOC (line of code) as a measure is used in many studies to predict the quality (2)(3), Complexity an internal property of software can be measured using different techniques applied to source code and design (4)(5). The common understanding about the complexity is its positive correlation with the defects. Although the relation is not always linear, it has significant impact. For the complexity measurement, different complexity metrics have been devised in past years (4)(5)(6)(7)(8). Studies showed that the majority of defects is caused by a small portion of the modules (3). These modules can be identified before time by examining the complexity to reduce the post release and maintainability work.

However it is not straightforward, for the complexity we have different complexity metrics. The selection of appropriate complexity metrics that best relate and indicate with the defects is of concern and requires minimal empirical evaluation for the selection.

In this paper, we aim to characterize and compare different complexity metrics for the defects based on different project categories in term of size. This characterization will help us to identify the complexity metrics with the best ability to indicate the defects in each project category. This observation will not only help us in indicating the complexity metrics having a positive correlation with defects, but also provide a breakthrough for the quality assurance techniques in accessing the projects.

The paper is organized as follows; Section 0 discussed the related work. Section **Error! Reference source not found.** presents the research design of the study. Section 3 presents the results. Finally, the conclusions are presented in Section.

2. RELATED WORK

Many studies show an acceptable correlation between complexity metrics and software defect proneness (9)(10)(11)(12).

English et al, highlighted the usefulness of the CK metrics for identifying the fault-prone classes (13). Gyimothy et al. studied the object oriented metrics given by CK for the detection of fault prone source code of open source Web and e-mail suite called Mozilla. They found that CBO metric seems to be the best in predicting the fault-proneness of classes and DIT metric is untrustworthy, and NOC cannot be used at all for fault-proneness prediction (12). Yu et al. examined the relationship between the different complexity metrics and the fault proneness. They used univariate analysis and found that WMC, LOC, CBOout, RFCout LCOM and NOC have a significant relationship with defects but CBOin, RFCin and DIT

have no significant relationship (14). Subramanyam and Krishnan examined the effect of the size along with the WMC, CBO and DIT values on the faults by using multivariate regression analysis for Java and C ++ classes. They conclude that size was a good predictor of defects in both languages, but WMC and CBO could be validated only for C++ (11).

Olague et al. studied three OO metrics suites for their ability to predict software quality in terms of fault-proneness: the Chidamber and Kemerer (CK) metrics, Abreu's Metrics for Object-Oriented Design (MOOD), and Bansiya and Davis' Quality Metrics for Object-Oriented Design (QMOOD). They concluded that CK and QMOOD suites contain similar components and produce statistical models that are effective in detecting error-prone classes. They also conclude that class components in the MOOD metrics suite are not good class fault-proneness predictors (15). However, Nagappan et al. stated that there is no single set of complexity metrics that could act as a universally best defect predictor (16).

From the related work we can extract different implications based on different studies, studying different projects about the relationship of complexity metrics and defect proneness. Hence the results are partial. To study and compare the behavior of complexity metrics indicating defect proneness, it requires an in depth study considering all the complexity metric measures and defects belonging to one project. In addition to generalize the results it requires to take into account many projects having all the metrics available and then analysis should be made collectively. In this way the actual relationship of all the complexity metrics with defect can be understood. Many complexity metrics have been studied in the previous researches (17)(15). Table 1 shows the metrics used in the study.

Table 1. The metrics used in the study

The metrics suggested by Chidamber and Kemerer (5) are.
Weighted Methods per class (WMC): WMC is the number of methods defined in each class.
Depth of Inheritance Tree (DIT): It is the measure of the number of ancestors of a class.
Number of Children (NOC): It is the measure of a number of direct descendants of the class.
Coupling between Objects (CBO): It is the number of classes coupled to a given class.
Response for a Class (RFC): It is the measure of different methods that can be executed when an object of that class receives a message.
Lack of Cohesion in Methods (LCOM): It is the number of pairs of member functions without shared instance variables, minus the number of pairs of member functions with shared instance variables.
Henderson Sellers defined one complexity metric (7).
Lack of cohesion in methods (LCOM3): According to study (7) LCOM3 is defined as. $$LCOM3 = \frac{\left(\frac{1}{a}\sum_{j=1}^{a}\mu(A_j)\right) - m}{1-m}$$ m - number of methods in a class; a - number of attributes in a class; μ(A) - number of methods that access the attribute A.

Bansiya and Davis (6) suggested the following quality metrics suite.
Number of Public Methods (NPM): It counts all methods in a class that are declared as public. This metric is also known as Class Interface Size (CIS).
Data Access Metric (DAM): It is the measure of the ratio of the number of private (protected) attributes to the total number of attributes declared in the class.
Measure of Aggregation (MOA): It is the count of the number of class fields whose types are user defined classes.
Measure of Functional Abstraction (MFA): It is the ratio of the number of methods inherited by a class to the total number of methods accessible by the member methods of the class.
Cohesion among Methods of Class (CAM): It computes the relatedness among methods of a class based upon the parameter list of the methods.
Data Access Metric (DAM): It is the measure of the ratio of the number of private (protected) attributes to the total number of attributes declared in the class.
Measure of Aggregation (MOA): It is the count of the number of class fields whose types are user defined classes.
Measure of Functional Abstraction (MFA): It is the ratio of the number of methods inherited by a class to the total number of methods accessible by the member methods of the class.
Cohesion among Methods of Class (CAM): It computes the relatedness among methods of a class based upon the parameter list of the methods.
Tang et al (18) extended the Chidamber & Kemrer metrics suite focusing on the quality.
Inheritance Coupling (IC): It provides the number of parent classes to which a given class is coupled.
Coupling Between Methods (CBM): It measures the number of new/redefined methods to which all the inherited methods are coupled.
Average Methods Complexity (AMC): It measures the average method size for each class.
Following two metrics were suggested by Martin (8).
Afferent Coupling (Ca): It is the number of classes that depend upon the measured class.
Efferent coupling (Ce): It presents the number of classes that the measured class is depended upon.
The one metric was suggested by McCabe (4).
McCabe's Cyclomatic Complexity (CC). It is equal to the number of different paths in a method (function) plus one. It is defined as CC = E-N+P; where E is the number of edges in the graph, N is the number of nodes of the graph; P is the number of connected components. It is only suitable for the methods; therefore it is converted to the class size metrics, by calculating the arithmetic mean of the CC value in the investigated class.

3. RESEARCH DESIGN

In this section, we present the research question and the data set. One research question is formulated for this research.

RQ1: Do complexity metrics have an effect on defects?

We selected the last releases of 38 software projects constituting 27,734 modules from the "*Promise data repository*[2]*"* having the required metrics freely available for research evaluation purposes (19). The data set contains 6 proprietary software projects, 15 open source software projects and 17 are academic software projects that were developed by the students. We downloaded the CVS files and found 18 complexity metrics defined in (Section 0). The values of complexity metrics were available against the defects for every module.

4. RESULTS

We carried out the preliminary analysis to identify the three categories of software projects small, medium and large using the K mean clustering algorithm. We found 24 software projects in the small category, 7 software projects in medium and 7 software projects in the large category. The average defects found in the small category of software projects are 52.5, for the medium category of software projects it is 519.14 and 508.2 defects for a large category of software projects. Table 2 shows the three categories of software projects i.e. small, medium and large.

Table 2. Categories of software's in term of size

Category	No	Avg defects	Avg size
Small [1-60KLoC]	24	52.5	17241 LoC
Medium [60–300 KLoC]	7	519.14	140743 LoC
Large [above 300KLoC]	7	508.2	427354 LoC

RQ 1: Do complexity metrics have an effect on defects?

We attempted to find the linear correlation between the complexity metrics and defects. We selected Pearson correlation coefficient which best suited to find the linear relation between the two variables. In no case we found the strong correlation among complexity metrics and defects.

To study any possible relationship of defects with complexity metrics, we cluster the modules into three categories based on the values, using the K mean clustering algorithm. In order to understand the clusters behavior, we performed the preliminary analysis of the identified clusters for each complexity metric based on the project category. We found three types of behaviors of complexity metrics and grade them as effective, untrustworthy and not useful indicators of defects.

Effective Indicators

We extract those complexity metrics where higher values result in higher defects. We called these complexity metrics effective indicators of defects and these metrics exhibit the phenomenon. Table 3 reports the complexity metrics, effective indicators of defects in small, medium and large projects.

[2] http://promisedata.org/

High Complexity → High Defect

Table 3 Complexity metrics effective indicators of defects

Project type	Complexity metrics
Small	LCOM
Medium	WMC, CBO, RFC, CA, CE, NPM, DAM, MOA, IC, Avg CC
Large	WMC, CBO, RFC, CA, NPM, AMC

Untrustworthy Indicators

We classify those complexity metrics that have no fixed criterion of increase in defect with the increase in complexity metric value. We called these complexity metrics untrustworthy indicators of defects. Table 4 reports the complexity metric untrustworthy indicator of defects in small, medium and large category of projects. For the untrustworthy indicators we observe two different behaviors of complexity metrics. (a) Medium complexity resulted in high defects: Medium Complexity → High Defects. (b) Large cluster values resulted in high defects but corresponding medium cluster value resulted in lower defects: High Complexity → High Defects, AND Medium Complexity → Low Defects.

Table 4 Complexity metrics untrustworthy indicators of defects

Project type	Complexity metrics
Small	WMC, NOC, CBO,RFC, CE, Avg CC
Medium	DIT, NOC, LCOM, CBM, AMC
Large	DIT, NOC, LCOM, CE, LCOM3, DAM, MOA, MFA, CAM, IC, Avg CC

Not useful indicators

We classify those complexity metrics where smaller values resulted in high defects, as not useful indicators of defects. Table 5 reports the complexity metrics not useful indicator of defects in small, medium and large projects. These complexity metrics exhibit the phenomenon**:** Low Complexity → High Defects

Table 5 Complexity metrics not useful indicators of defects

Project type	Complexity metrics
Small	WMC, DIT, CA, NPM, LCOM3, DAM, MOA, CAM, IC, CBM, AMC
Medium	LCOM3, MFA, CAM
Large	CBM

Hypothesis testing

For hypothesis testing, we only consider the effective indicator of complexity to verify that the distribution of defects among high, medium and low complexity. We did not perform the analysis on the untrustworthy and not useful indicators because it does not seem to be very meaningful. We take support of statistical hypothesis testing to confirm the difference of defect in three categories of complexi-

ty metrics i.e. high, medium and low. Using statistical techniques, we will test the null hypothesis H_0. We will accept and reject it based on the favor of the alternative hypothesis.

- H_0: There is no significant difference of defects among high, medium and low complexity of effective indicators.
- H_1: There is a significant difference of defects among high, medium and low complexity of effective indicators.

Selection of statistical test

We first examined the distribution of the samples to choose the appropriate statistical test for the analysis. We applied the Ryan-Joiner test for the normality check and found that for every sample (p - value <0.01). The results showed that none of the sample under study has a normal distribution of data. This made us to select the non parametric test for the hypothesis testing.

According to the recommendation for not normal samples, we chose non parametric test Kruskal - Wallis test for differences between three or more samples. We compare the defects of high, medium and low complexity cluster of each effective metric. In each category of projects the obtained p value was found less than 0.05 meaning there is a significant difference of defects among high, medium and low complexity value of effective indicators.

4.1 Threats To Validity

This section discusses the validity threat as classified and proposed by study (20). As for construct validity, we collected the CVS logs from the Promise research data repository. Although we have much confidence in the correctness and accuracy of the provided data but still we have no control to decide that up to which level the data is authentic e.g. how many module's data may be left to record, correctness of the measurement of complexity metric values etc.

As for external validity, our findings are based on the large data set of modules i.e. 27,734 of 38 software projects. Although this number is not small but still there can raise some concerns on the generalization of the findings when the projects are categorized into small, medium and large. We have 24 projects from small, 7 projects are from medium and 7 projects are from large category which are fairly small samples.

5. CONCLUSIONS

The findings have important implications as they are based on the complete set of complexity metrics belonging to one particular project. Similarly 38 such projects were selected which have all the complexity metrics available and then combined to perform the analysis collectively. The artifact of this study is very vital

and beneficial for both researchers and practitioners. The primary contribution of this research is the identification of having no linear relation of any complexity metrics with defects. However based on the complexity metrics high, medium and small value clusters we find that there are some complexity metrics which higher values resulted in a higher number of defects. These complexity metrics are called effective indicators of defects. Consequently the complexity has an effect on defects but not as large as one might expect. The researchers can use the effective complexity metrics for the predicting and estimating of defects and can use in predictive models. The statistical analysis adds confidence that there is a quite significant difference among the defects of effective complexity metrics high, medium and low values. The practitioners can assess their projects' quality based on the effective complexity metrics. The categorization of projects is quite useful as it gives practitioners a view to select the appropriate effective complexity metric when assessing their project based on size. We would like to continue our future studies with the same attention considering number of projects to validate the artifacts of this study and thus generalize the effective complexity metrics for small, medium and large projects.

6. REFERENCES

1. Güneş Koru A, Tian J. An empirical comparison and characterization of high defect and high complexity modules. Journal of Systems and Software. 2003 Sep 15;67(3):153–63.
2. Hongyu Zhang. An investigation of the relationships between lines of code and defects. Software Maintenance, 2009. ICSM 2009. IEEE International Conference on. 2009. p. 274–83.
3. Fenton NE, Ohlsson N. Quantitative analysis of faults and failures in a complex software system. Software Engineering, IEEE Transactions on DOI - 10.1109/32.879815. 2000;26(8):797–814.
4. McCabe TJ. A Complexity Measure. Software Engineering, IEEE Transactions on DOI - 10.1109/TSE.1976.233837. 1976;SE-2(4):308–20.
5. Chidamber SR, Kemerer CF. A metrics suite for object oriented design. Software Engineering, IEEE Transactions on DOI - 10.1109/32.295895. 1994;20(6):476–93.
6. Bansiya J, Davis CG. A hierarchical model for object-oriented design quality assessment. Software Engineering, IEEE Transactions on DOI - 10.1109/32.979986. 2002;28(1):4–17.
7. Henderson-Sellers B. Object-Oriented Metrics, measures of Complexity. Prentice Hall; 1996.
8. Martin R. OO Design Quality Metrics - An Analysis of Dependencies.
9. Chidamber SR, Darcy DP, Kemerer CF. Managerial use of metrics for object-oriented software: an exploratory analysis. Software Engineering, IEEE Transactions on DOI - 10.1109/32.707698. 1998;24(8):629–39.
10. Basili VR, Briand LC, Melo WL. A validation of object-oriented design metrics as quality indicators. Software Engineering, IEEE Transactions on DOI - 10.1109/32.544352. 1996;22(10):751–61.

11. Subramanyam R, Krishnan MS. Empirical analysis of CK metrics for object-oriented design complexity: implications for software defects. Software Engineering, IEEE Transactions on DOI - 10.1109/TSE.2003.1191795. 2003;29(4):297–310.
12. Gyimothy T, Ferenc R, Siket I. Empirical validation of object-oriented metrics on open source software for fault prediction. Software Engineering, IEEE Transactions on DOI - 10.1109/TSE.2005.112. 2005;31(10):897–910.
13. English M, Exton C, Rigon I, Cleary B. Fault detection and prediction in an open-source software project. Proceedings of the 5th International Conference on Predictor Models in Software Engineering. Vancouver, British Columbia, Canada: ACM; 2009. p. 1–11.
14. Ping Yu, Systa T, Muller H. Predicting fault-proneness using OO metrics. An industrial case study. Software Maintenance and Reengineering, 2002. Proceedings. Sixth European Conference on. 2002. p. 99–107.
15. Olague HM, Etzkorn LH, Gholston S, Quattlebaum S. Empirical Validation of Three Software Metrics Suites to Predict Fault-Proneness of Object-Oriented Classes Developed Using Highly Iterative or Agile Software Development Processes. Software Engineering, IEEE Transactions on DOI - 10.1109/TSE.2007.1015. 2007;33(6):402–19.
16. Nagappan N, Ball T, Zeller A. Mining metrics to predict component failures. Proceedings of the 28th international conference on Software engineering. Shanghai, China: ACM; 2006. p. 452–61.
17. Catal C, Diri B, Ozumut B. An Artificial Immune System Approach for Fault Prediction in Object-Oriented Software. Dependability of Computer Systems, 2007. DepCoS-RELCOMEX ’07. 2nd International Conference on. 2007. p. 238–45.
18. Mei-Huei Tang, Ming-Hung Kao, Mei-Hwa Chen. An empirical study on object-oriented metrics. Software Metrics Symposium, 1999. Proceedings. Sixth International. 1999. p. 242–9.
19. Boetticher G, Menzies T, Ostrand T. PROMISE Repository of empirical software engineering data [Internet]. Available from: http://promisedata.org/ repository, West Virginia University, Department of Computer Science, 2007
20. Wohlin C, Runeson P, Höst M, Ohlsson MC, Regnell B, Wesslen A. Experimentation in software engineering: an introduction. Norwell, Massachusetts. USA: Kluwer Academic Publishers; 2000.