Large Method, Class and long Parameter list detection using Traditional, CK and Mood metric on OOP python project

Mahir Mahbub
Institute of Information Technology
University of Dhaka
Dhaka, Bangladesh
bsse0807@iit.du.ac.bd

Hasan Tarek
Institute of Information Technology
University of Dhaka
Dhaka, Bangladesh
bsse0818@iit.du.ac.bd

S. M. Khayrul Islam
Institute of Information Technology
University of Dhaka
Dhaka, Bangladesh
bsse0822@iit.du.ac.bd

Obaidur Rahman
Institute of Information Technology
University of Dhaka
Dhaka, Bangladesh
bsse0830@iit.du.ac.bd

Md. Saeed Siddik
Institute of Information Technology
University of Dhaka
Dhaka, Bangladesh
saeed.siddik@iit.du.ac.bd

***Abstract*—Software development plays an important role in the delivery and application of information technology. The focus on assessing the quality of these software are rapidly growing. Software metrics plays vital role assessing the quality of software measures on object oriented software. This paper describes the result of an investigations in a set of metrics of two python object oriented projects. The metrics suits are traditional, CK and MOOD metrics. It also focuses on detecting large class, large method and long parameter list code smells using the measured metrics.**

***Index Terms*—Software Metrics, MOOD, CK, Code smell**

I. Introduction

Several years ago, software considered as a technical business where the key of success was the functionality. Nowadays functionality is not sufficient but quality with functionality is a critical issue for success. For excellent product the quality of the product must be well defined and measured. So, how do we measure the quality of a software is a major concern. To measure the quality of a software metrics we need to calculate the various metrics of a software [1].

Modularity and reusability are the main advantage of object-oriented design. Now a day's software projects are shifting to OO (object oriented) design because of its development process is different from structured program development process. Object oriented designed software program metrics are measured by object-oriented metrics. Metrics describe complexity, size and robustness and keep track of design performance of object-oriented projects are called designed based metrics. To obtain accurate estimation of project milestones and contains minimal faults, metrics are a means of. Object oriented design is new technology compared to structured development. Metrics have not same effect on both structured and object-oriented design project. "Line of code" metric used in structured development much more whereas it used in object-oriented design poorly. Corrective maintenance cost saving of 42% by using object-oriented metrics.

This paper is organized in the following way. Next section discusses the background study, section 3 describes the dataset that is used to evaluate the metrics, section 4 focuses on how we have calculated different metrics for evaluation. Section 5 and 6 discusses about the result and findings from the result consecutively.

II. Background

Although the Object-oriented metrics are newer in the measurement theory, they have proven as useful predictors of good system design. A wide variety of object-oriented metrics have been proposed to assess

the testability of an object-oriented system.

From day to day interest in maintaining software quality is growing. Metrics are standard measurement to assess the quality of a software [2]. A number of software metrics have been proposed and some are still proposing. Some metrics were proposed for some specific programming language. For object oriented programming languages there also have been proposed some software metrics. There is a large number of metrics that can be calculated for a software project but which metrics meet the needs of a software project is a major concern [3].

In literature a significant number of object oriented metrics have been developed and some are still developing. Traditional metrics model is a metrics design model which is applied on the method of a class.A Traditional software metric is McCabe Cyclomatic Complexity (CC) from which it can be inferred the constancy and maintainability of a software system by analysing source code. It also contains Source Lines of Code(SLOC). SLOC is used to measure the size of a software program by counting the number of lines in the program's source code. It is used to predict the amount of effort that will be required to develop a program [4]. The comment percentage (CP) metric is calculated by the total number of comments divided by the total lines of code less the number of blank lines.

To assess the object oriented software numerous metrics can be calculated. Metrics proposed by CK metrics [5] Abreu [2], Li and Henry [6], Lorenz and Kidd [7] metrics, MOOD metrics [7] etcs. are mostly used metrics among them.

Chidamber and Kemerar proposed six metrics which are known as C.K metrics. These metrics are Weighted Method per Class (WMC), Depth of Inheritance Tree (DIT), Number of Children (NOC), Coupling between Objects (CBO), Response for a Class (RFC) and Lack of Cohesion in Methods (LCOM) [4]. WMC measures the complexity of a class, it indicate the required effort to maintain a particular class. DIT is the measurement of the maximum path length from the node to root of the tree. NOC counts the number of sub-classes that inherits the parent class. A class is coupled with another if the methods of one class use the methods or attributes of the other class. An increase of CBO indicates the reusability of a class will decrease. So, CBO values for each class should be kept as low as Possible. LCOM uses the notion of degree of how much similarity exists in methods.

Abreu [8] defined Metrics for Object Oriented Design (MOOD) metrics. MOOD metrics refers encapsulation, inheritance, polymorphism in object oriented paradigm. Two main features are used in MOOD metrics; they are methods and attributes. MOOD metrics include Method Hiding Factor (MHF), Attribute Hiding Factor (AHF), Method Inheritance Factor (MIF), Attribute Inheritance Factor (AIF). Polymorphism Factor (POF) and Coupling Factor (COF). Most of the metrics focus on encapsulation, inheritance, class complexity and polymorphism. CK metrics suite is a set of six metrics which capture different aspects of an OO design; these metrics mainly focus on the class and the class hierarchy. It includes complexity, coupling and cohesion as well. On the other hand, MOOD metrics focus on system level which includes encapsulation, inheritance, polymorphism, and massage passing. Software metrics have been traditionally used to evaluate the maintainability of the software systems and to detect code smells. Code smells are symptoms that may indicate something wrong in the system code.

III. Dataset Description

To calculate the metrics we search over the internet and selected two python object oriented projects. These projects name and online link is given below:

SI	Project Name	Github Link
1	ATM System	https://github.com/MahirMahbub/ATM
2	Python OOP Toy	https://github.com/LambdaSchool/Python-OOP-Toy

IV. Methodology

In this paper we have calculated traditional, CK and MOOD metrics to assess the quality of a software and using these calculated metric detect one or more code smell if anyone exists in the software. The following metrics have been calculated

Traditional Metrics

- McCabe Cyclomatric Complexity
- Line of Code (LOC)
- Source Lines of Code (SLOC)
- Commented Lines of Code (CLOC)
- Comment Percentage

To calculate cyclomatic complexity we constructed graph with nodes and edges from the code, then identify the independent paths and finally calculated the cyclomatic complexity of the code. LOC, SLOC, CLOC is calculated reading the source code of the project. The comment percentage (CP) is calculated by the total number of comments divided by the total lines of code less the number of blank lines
Comment percentage = total number of comments / (total lines of code - the number of blank lines)

CK Metrics

- Weighted method per class (WMC)
- Number of children (NOC)
- Coupling between objects (CBO)
- Lack of cohesion in methods (LCOM)

WMC measures the complexity of a class. To calculate WMC we calculated the complexity of all methods of a class and the sum it. The lower the WMC value the better for a class. DIIT is calculated by generating a graph from parent class to all of its sub-classes and the maximum length of all graph path is the DIT. DIT represents the complexity of the behavior of a class, the complexity of design of a class and potential reuse of a class. From this generated graph we also calculated the number of children of a parent class. The size of NOC indicates the level of reuse of o class in an application. CBO is calculated by checking if a class's method use the method of another class's attribute or method.

Mood Metrics

- Method hiding factor (MHF)
- Attribute hiding factor(AHF)
- Method inheritance factor (MIF)
- Coupling factor (COF)

The MHF metric indicates the sum of the invisibilities of all methods in all classes. It is calculated as

$$MHF = \sum_{i=1}^{TC} M_h(C_i) / \sum_{i=1}^{TC} M_d(C_i) \tag{1}$$

Here, $M_d(C_i) = M_v(C_i) + M_h(C_i)$
$M_d(C_i) =$ the number of methods defined in class C_i
$M_v(C_i) =$ the number of methods that visible in the class C_i
$M_h(C_i) =$ the number of methods hidden in C_i

The AHF metric shows the sum of the invisibilities of all attributes in all classes. The MHF metric is defined as follows:

$$AHF = \sum_{i=1}^{TC} A_h(C_i) / \sum_{i=1}^{TC} A_d(C_i) \tag{2}$$

Here, $A_d(C_i) = A_v(C_i) + A_h(C_i)$
$A_d(C_i) =$ the number of attributes defined in class C_i
$A_v(C_i) =$ the number of attributes that visible in the class C_i
$A_h(C_i) =$ the number of attributes hidden in C_i

In python attribute there are three types of variant, these are attribute, normal variable and tuple variable. We extracted all this variable using Abstract Syntax Tree (AST) with nested object checking. Then calculated the AHF value.
The MIF metric defined as the sum of inherited methods in all classes of the system. It also indicates the degree to which the class architecture of an object oriented system makes use of inheritance for both methods and attributes. MIF is calculated by using the following formula.

MIF is defined as the ratio of the sum of the inherited methods in all classes of the system as follows:

$$MIF = \sum_{i=1}^{TC} M_i(C_i) / \sum_{i=1}^{TC} M_a(C_i) \tag{3}$$

Here, $M_a(C_i) = M_d(C_i) + M_i(C_i)$
$M_a(C_i) =$ the number of methods defined in class C_i
$M_d(C_i) =$ the number of methods declared in the class C_i
$M_i(C_i) =$ the number of methods inherited in C_i

The COF is defined as the ratio of the maximum possible number of couplings in the system to the actual number of coupling is not imputable to inheritance. The COF is defined as follows:

$$COF = \sum_{i=1}^{TC} \left[\sum_{j=1}^{TC} is\_Client(C_i, C_j) \right] / (TC^2 - TC^2) \tag{4}$$

Here, $is\_Client(C_i, C_j) = 1$ if and only if, a relationship exists between the Cc and the server class Cs and Cc not equal to Cs.
And, $is\_Client(C_i, C_j) = 0$, otherwise

COF and CBO both uses the idea of coupling factor. The main difference between COF and CBO is, in COF metric all variable accesses are counted but in CBO metric variable count is omitted.

V. Results

Traditional, CK and MOOD metric results for the selected projects are given below:

Table I
Traditional Metrics

Project Name	LOC	LC	SLC	CP
ATM System	327	3	17	0.061
Python OOP Toy	114	0	17	0.17

Table II
CK Metrics

Project Name	DIT
ATM System	2
Python OOP Toy	1

Table III
MOOD Metrics

Project Name	MHF	AHF	MIF	COF
ATM System	0.267	0	0.16	1
Python OOP Toy	0.292	0	0	0

The data acquired from the metrics calculation scheme also analysed and plotted for further analysis and to show the findings:

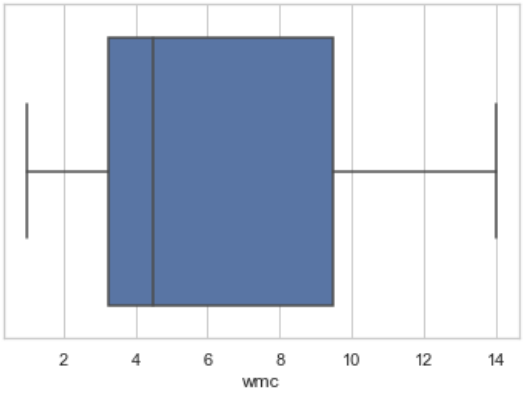

Figure 1. Box plot of Weighted Method per Class

Table IV
Correlation Matrix of WMC and LOC

	WMC	LOC
WMC	1	0.933
LOC	0.933	1

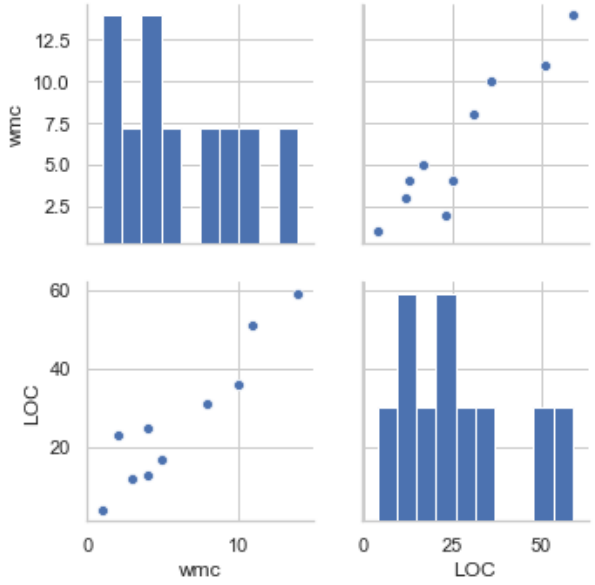

Figure 2. Correlation among LOC of Method, Parameter Length and Method complexity

From figure 2 there exists a positive correlation between WMC and LOC. Table 2 describes correlation between these two metrics and from the value we can see they are strongly correlated to each other.

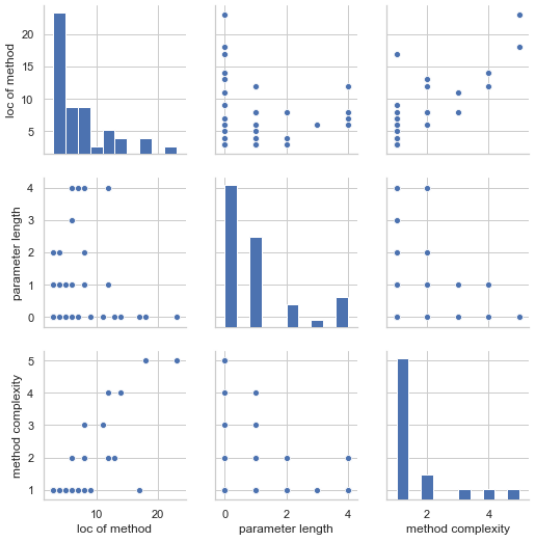

Figure 3. Correlation between WMC and LOC

Table V
CORRELATION METRICS FOR MLOC, PARAM LENGTH AND MC

	LOC of Method	**Parameter Length**	**Method Complexity**
LOC of Method	1	-0.048	0.80
Parameter Length	-0.048	1	-0.16
Method Complexity	0.80	-0.16	1

From figure 3 and table 2 it is shown that there is a positive correlation of value 0.80 between MLOC and MC, there exist a negative correlation between parameter length and MC of -0.16 which is negligible. There is almost no relation between parameter length and MLOC.

VI. FINDINGS

A class in python will be called a large class if it has LOC greater than or equal 200 or the number of attribute and method is greater than 40 [9]. Long method can be detected if the method lines of code is greater than or equal 100 and if the parameter list of a method is greater or equal 5.

From the result section it has been seen that there is a strongly positive correlation between WMC and LOC of class, we can say that if the WMC value is much bigger there is a high probability of a class being a god class.

It is also true for large method as we can have seen a positive correlation MLOC and MC. MC and parameter length is negatively correlated though the value is small but there is somehow a negative relation that tends to a complex method having less number of parameter.

The projects that we have selected for test do not have any class that exceeds the LOC of 200, parameter list greater or equal 5 and method of LOC greater than 100, that refers these code smells do not exists in these projects.

VII. CONCLUSION

We calculated above metrics related to traditional, CK and MOOD metrics. Using this metric we tried to detect large class, large method and long parameter list code smells. To detect this code smell of a project the project should be written by following object oriented paradigm and with pure python code. Otherwise the detection of the code smell may result bad interpretation of the project or the smell detection tool cause failure. In future, further code metrics calculation and using these metrics we will try to detect other code smells.

REFERENCES

[1] Johny P Antony. Predicting reliability of software using thresholds of ck metrics. *International Journal of Advanced Networking and Applications*, 4(6):1778, 2013.

[2] Fernando Brito e Abreu and Rogério Carapuça. Candidate metrics for object-oriented software within a taxonomy framework. *Journal of Systems and Software*, 26(1):87–96, 1994.

[3] C Neelamegam and M Punithavalli. A survey-object oriented quality metrics. *Global Journal of Computer Science and Technology*, 9(4):183–186, 2009.

[4] Kaushal Bhatt, Vinit Tarey, Pushpraj Patel, Kaushal Bhatt Mits, and Datana Ujjain. Analysis of source lines of code (sloc) metric. *International Journal of Emerging Technology and Advanced Engineering*, 2(5):150–154, 2012.

[5] Shyam R Chidamber and Chris F Kemerer. A metrics suite for object oriented design. *IEEE Transactions on software engineering*, 20(6):476–493, 1994.

[6] Wei Li and Sallie Henry. Maintenance metrics for the object oriented paradigm. In *[1993] Proceedings First International Software Metrics Symposium*, pages 52–60. IEEE, 1993.

[7] Rachel Harrison, Steve J Counsell, and Reuben V Nithi. An evaluation of the mood set of object-oriented software metrics. *IEEE Transactions on Software Engineering*, 24(6):491–496, 1998.

[8] F Brito e Abreu. The mood metrics set. In *proc. ECOOP*, volume 95, page 267, 1995.

[9] Zhifei Chen, Lin Chen, Wanwangying Ma, and Baowen Xu. Detecting code smells in python programs. In *2016 International Conference on Software Analysis, Testing and Evolution (SATE)*, pages 18–23. IEEE, 2016.