IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

EVALUATE CODING EFFICIENCY USING OBJECT ORIENTED METRICS AND COCOMO-III METHODOLOGIES. IMPROVE SOFTWARE FLAWS: A REVIEW

Neeta Mourya, Shanu K Rakesh Department of computer science and Engineering, Chouksey Engineering College, Bilaspur Chhattisgarh 495004, INDIA

***Abstract*:** Object-oriented programming is a programming methodology related to concepts such as class, object, and inheritance. Encapsulation, abstraction, and polymorphism. Popular programming languages such as Java, C++, C#, and Ruby use an object-oriented programming paradigm. Object-oriented metrics assess the quality of a design by measuring its class and attributes.This is closely related to the quality of the software. Databases play crucial roles in projects and applications. During software testing, testers find errors by comparing actual and predicted output. Software testing identifies faults and ensures error-free functionality for clients.

Using COCOMO-III and MOOD metrics, I suggest improving coding efficiency and accuracy to improve project evaluation findings. These results are useful for improving software estimation, quality training, and research.

***Index Terms* - Object oriented, Mood metrics, CK metrics, COCOMO model, Defects, measurement, coding etc.**

I. INTRODUCTION

Software quality is crucial for the development of software systems, particularly large-scale ones. High-quality software reduces maintenance costs and increases possibility for reuse. Software metrics can provide a quantitative and objective assessment of software quality.

Software measuring has become increasingly important in software engineering. Successful software developers measure software attributes to ensure consistent and full requirements, high-quality design, and tested code. Effective project managers evaluate process and product attributes to determine when software is ready for delivery or if budget has been exceeded. Target customers measure aspects of the final product to determine if its quality meets their requirements. Maintenance staff must be able to assess the current product to see what should be upgraded and improved. The metrics for object-oriented design focus on measurements that are applied to the class and design characteristics. These measurements permit designers to access the software early in process, making changes that will reduce complexity and improve the continuing capability of the design.

There are many object-oriented metrics models available and several authors have proposed ways to measure object-oriented design. The motivation of this thesis is to give an overview to Increase the Correctness and accuracy of how to improve the coding efficiency of an impact the results when evaluating the project using object-oriented approaches.

II. OBJECT ORIENTED DESIGN

Object-Oriented Design (OOD) is a programming approach that involves planning a system of interacting objects for the purpose of solving a software problem. It leverages the concepts of objects and classes, encapsulation, inheritance, and polymorphism to model real-world entities and their interactions. OOD is used to create a system architecture that is modular, flexible, and easy to understand and maintain.

III. INTERNAL QUALITY OF OOD

Internal quality of Object-Oriented Design (OOD) refers to characteristics like maintainability, readability, and testability. Here are some other things to consider when evaluating the quality of OOD:

- Inheritance: While inheritance is useful for reusing code, it can go against the goal of decoupled classes.
- Composition: Composition is another way to achieve code reuse.
- Type versus class: A type is an interface, which is a collection of methods that an object responds to.

- Design for change: Patterns can help address changes that might otherwise require redesign.
- Application framework: An application framework is a set of libraries or classes that can be used to implement the standard structure of an application. This can save time for developers by reducing the amount of code they need to rewrite for each new application.
- Persistent objects: Identify objects that need to last longer than a single application runtime.
- Remote objects: Identify and define remote objects and their variations.

IV. MOOD METRICS

Application quality is critical to the development of software systems, especially large-scale ones. High quality software would reduce the cost of software maintenance, and it enhances the potential software reuse.

In order to measure the software quality more quantitatively and objectively, software metrics (MOOD) give impression to be a powerful and effective methodology that decide a grade to an object-oriented application. So, in this section, we will discuss the mood factors to assess an object-oriented application.

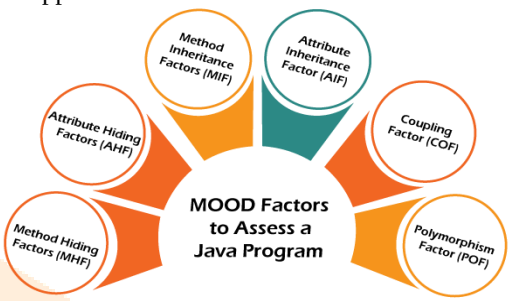

Fig 4.1 mood metrics

MOOD stands for Matrices for Object Oriented Design. It describes the characteristics of an object-oriented application or program. It is used to evaluate a grade of Java program. MOOD factors include the following six software quality indicators or metrices:

- Method Hiding Factor (MHF)
- Attribute Hiding Factor (AHF)
- Method Inheritance Factor (MIF)
- Attribute Inheritance Factor (AIF)
- Coupling Factor (COF)
- Polymorphism Factor (POF)

V. CK METRICS

The CK Metrics Suite comprises six metrics, each providing insights into different aspects of software design and implementation. These metrics serve as invaluable indicators for various dimensions of software quality:

5.1 Weighted Methods per Class (WMC):

- Measures the complexity of a class by assessing the sum of complexities of its methods.
- Identifies potential code smells and helps in evaluating maintainability.

5.2 Depth of Inheritance Tree (DIT):

- Indicates the maximum length from the node to the root of the inheritance tree.
- Offers insights into the hierarchical structure of the classes and potential complexity.

5.3 Number Of Children (NOC):

- Represents the number of immediate subclasses a class has.
- Provides an understanding of class reuse and potential dependencies.

5.4 Coupling Between Object Classes (CBO):

- Measures the number of classes to which a class is coupled.
- Helps in evaluating the level of interdependence among classes.

5.5 Response For a Class (RFC):

- Counts the number of methods that can potentially be executed in response to a message received by an object of the class.
- Highlights the potential interactions and responsibilities of a class.

5.6 Lack of Cohesion in Methods (LCOM):

- Measures the lack of cohesion among methods in a class.
- Indicates how closely related or unrelated the methods within a class are.

V. MOOD

The MOOD metrics, defined by Fernando Brito e Abreu, are designed to provide a summary of the overall quality of an object-oriented project. The original MOOD metrics suite consists of 6 metrics. The MOOD2 metrics were added later.

These metrics are useful for projects with heavy use of object-oriented programming. If your project relies mostly on standard modules and classic .frm files, the object-oriented metrics are not a good choice for you, even if your project contains some classes. The MOOD/MOOD2 metrics are available at the project level. They describe the entire project, not an individual class.

When counting the MOOD/MOOD2 metrics, Project Analyzer uses the following definitions.

Class
Classic VB: .cls or .ctl file
VB.NET: Class..End Class
Method (operation)
Classic VB: A sub, function or property procedure.
VB.NET: A sub, function, property accessor or operator.
Attribute (variable)
Classic VB: A variable or array defined in the declarations section of a class.
VB.NET: A class-level variable or array. It can be defined Shared or not. Please notice that MOOD attributes have nothing to do with the .NET <attributes>.

VII. PROGRAMMER CODING EVALUATION AND PERFORMANCE MEASUREMENT

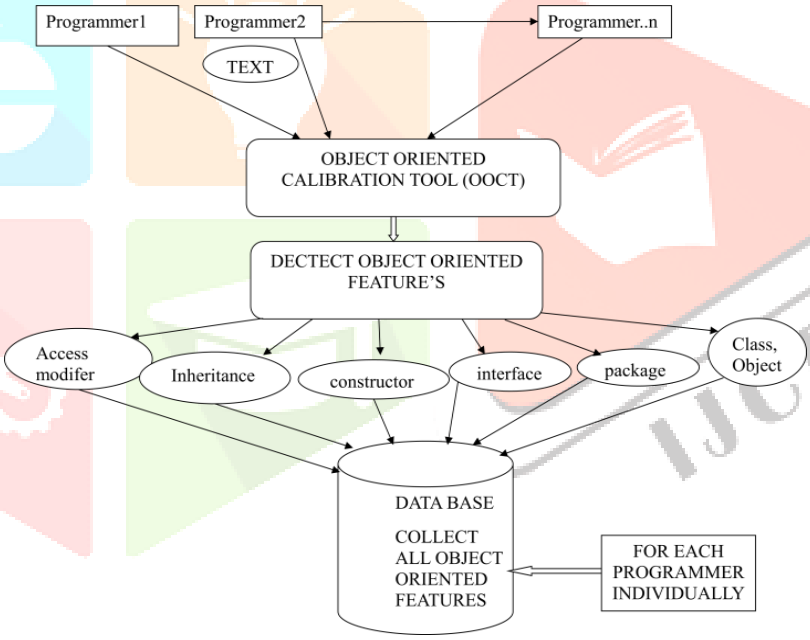

Fig 7.1 Programmer coding key evaluation

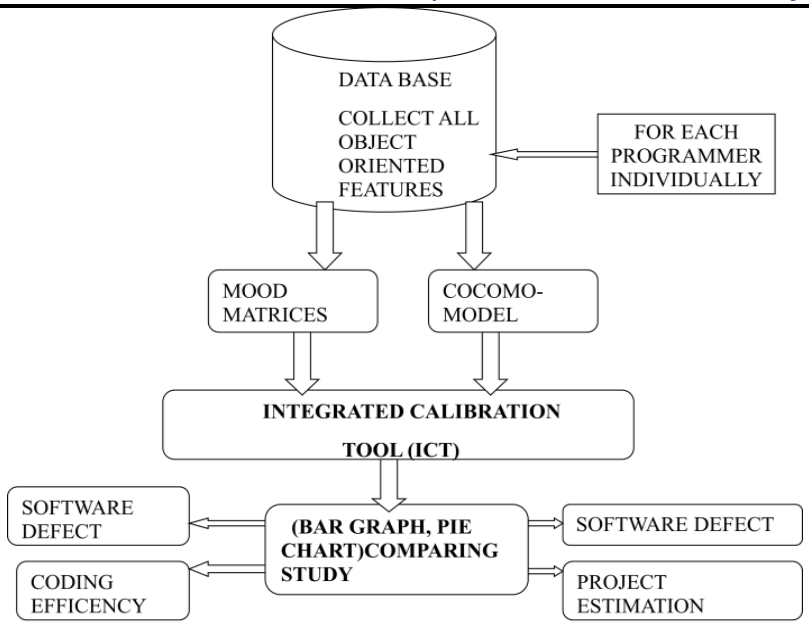

Fig 7.2 Coding evaluation & performance measurement.

VIII. FACTOR CALCULATION

8.1 Method Hiding Factor (MHF)

MHF can be calculates by using the following mathematical formula:

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

Where,
$M_h(C_i)$ = hidden Methods in class C_i
$M_d(C_i) = M_v(C_i) + M_h(C_i)$: Methods defined in C_i
$M_v(Ci)$: visible Methods in class C_i
TC: Total number of Classes

8.2 Attribute Hiding Factors (AHF)

AHF can be calculates by using the following mathematical formula:

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

Where,
$A_h(C_i)$ = hidden attributes in class C_i
$A_d(C_i) = A_v(C_i) + A_h(C_i)$: attributes defined in C_i
$A_v(Ci)$: visible attributes in class C_i
TC: Total number of classes

8.3 Inheritance Factor (MIF)

MIF can be calculated by using the following mathematical formula:

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

Where,
M_i: inherited methods
$M_a(C_i)$ = $M_d(C_i) + M_i(C_i)$: attributes defined in C_i
$M_d(Ci)$: defined methods
TC: Total number of classes
At first sight, we might be tempted to think that inheritance should be used extensively. However, the composition of several inheritance relations builds a directed acyclic graph (inheritance hierarchy tree), whose depth and width make understandability and testability fade away quickly [1,2,3,4].

8.4 Attribute Inheritance Factor (AIF)

AIF can be calculated by using the following mathematical formula:

$$AIF = \frac{\sum_{i=1}^{TC} A_i(C_i)}{\sum_{i=1}^{TC} A_a(C_i)}$$

Where,
$A_h(C_i)$ = hidden attributes in class C_i
$A_d(C_i)$ = $A_v(C_i) + A_h(C_i)$: attributes defined in C_i
$A_v(Ci)$: visible attributes in class C_i
TC: Total number of classes

8.5 Coupling Factor (COF)

It measures the coupling between classes.
COF can be calculated by using the following mathematical formula:

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

Where,
is_client(C_c,C_s) =| 1 if ($C_c \Rightarrow C_s$)^($C_c \neq C_s$) , 0 otherwise
TC: It denotes the total number of classes.

8.6 Polymorphism Factor (POF)

Polymorphism means having the ability to take several forms. In OO systems, polymorphism allows the implementation of a given operation to be dependent on the object that "contains" the operation 2, 3, 4, and 6.

$$POF = \frac{\sum_{i=1}^{TC} M_o(C_i)}{\sum_{i=1}^{TC}\left[M_n(C_i) \times DC(C_i)\right]}$$

Where,
$M_o(C_i)$: overriding Methods in class C_i
$M_n(Ci)$: new Methods in class C_i
$D_C(C_i)$: number of Descendants of Class C_i (derived classes)
TC: Total number of Classes

IX. The Oo Evaluating

The grade awarded to an object-oriented program depends on the following table that is based on experiments.

Table 9.1: The range of mood factors

Factor	Minimum	Maximum	Minimum Tolerance	Maximum Tolerance
MHF	12.7%	21.8%	9.5%	36.9%
AHF	75.2%	100%	67.7%	100%
MIF	66.4%	78.5%	60.9%——	84.4%
AIF	52.7%	66.3%	37.4%	75.7%
COF	0%	11.2%	0%	24.3%
POF	2.7%	9.6%	1.7%	15.1%

We observe that each MOOD factor is associated with basic structural mechanisms of the object-oriented model. It is the mechanism that enables some recommendations for designers. The interval of each MOOD factor has been altered that is based on experimental results, to be fit in the evaluation of Java Programs. The weight factor reflects the importance of each characteristic.

X. PROPOSED ALGORITHM

Step l. Enter text (program) step for n to l.
Step 2. Collect and store text in Object Oriented Calibration Tool (OCT), oriented calibration tool is tool for storing the object-oriented database.
Step 3. Detect Object Oriented Features of text.
Step 4. Collect and Store all Object-Oriented features in Database for each programmer Individually.
Step 5. Apply Mood Metrics, CK Metrics and COCOMO III Model.
Step 6. According to database produce following result.

- Using ICT (Integrated Calibration Tool) generate Bar, Pie, Line Chart and comparing Study table.
- Produce coding efficiency
- Produce Project Estimation.

Step 7. End.

XI.CONCLUSION

• The system evaluates several program attributes based on complexity and design. In each example, the system correctly discovered the vulnerabilities in the evaluated program.
• Using these data can improve software estimating, quality training, and project milestone estimation accuracy.
• Creating a software system with minimum errors.
• To improve software quality, employ inheritance and simplify the class structure.

REFERENCES

[1] Ganesh Chandra, D. L. Gupta, 3A. K. MalviyaSome "Observations based on Comparison of MOOD and CK Software Metrics suites for Object Oriented System" IJCST July - Sept 2012ISSN : 0976-8491, Vol. 3, Issue 3.
[2] Jaechang Nam , Wei Fu, Student Member, IEEE, Sunghun Kim, Member, IEEE, Tim Menzies , Member, IEEE, and Lin Tan, Member, IEEE "Heterogeneous Defect Prediction" IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 44, NO. 9, SEPTEMBER 2018.
[3] Zhiyuan Wan, Xin Xia, Ahmed E. Hassan, David Lo, Jianwei Yin, and Xiaohu Yang "Perceptions, Expectations, and Challenges in Defect Prediction" IEEE TRANSACTIONS ON SOFTWARE ENGINEERING.
[4] Ping Cao a , Ke Yang b , ∗, Ke Liu c , d "Optimal selection and release problem in software testing process: A continuous time stochastic control approach" European Journal of Operational Research March 2, 2019.
[5] Tirimula Rao Benalaa,∗, Rajib Mallb "DABE: Differential evolution in analogy-based software development effort estimation"Swarm and Evolutionary Computation 38 (2018) 158–172.
[6] Kuljit Kaur and Hardeep Singh Deptt. Of Computer Science and Engineering, Guru Nanak Dev University, Amritsar "Analyzing Software Evolution Using the MOOD Metric Set", IJARCS,ISSN NO.0976-5697,VOL2,NO.1,Feb 2011.
[7] Alexander Egyed, Member, IEEE "Automatically Detecting and Tracking Inconsistencies in Software Design Models" IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 37, NO. 2, MARCH/APRIL 2011.
[8] Ning Nan and Donald E. Harter, Member, IEEE "Impact of Budget and Schedule Pressure on Software Development Cycle Time and Effort" IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 35, NO. 5, SEPTEMBER/OCTOBER 2009.
[9] Garima Verma Master of Computer Application Dept. Dehradun Institute of Technology, Dehradun "Software Defects and Object Oriented Metrics - An Empirical Analysis" International Journal of Computer Applications (0975 – 8887) Volume 9– No.5, November 2010.
[10] Manish Agrawal and Kaushal Chari "Software Effort, Quality, and Cycle Time: A Study of CMM Level 5 Projects" IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 33, NO. 3, MARCH 2007.