Optimization approach to improving coding efficiency based on Object-oriented metrics and COCOMO-II technique

[1]Dr. Sudhir Kumar Meesala, [2]Ragini yadav

[1]Department of CSE, Assistant professor, Chouksey engineering college, India.

Abstract: **With the increasing size and complexity of software's, Software development has become a more clamorous process and hence needs to take care of even the simplest activity in the development process. The problems being faced in the software developments are quality degradation, cost overrun, schedule overrun.**

Measurement programs in software organizations are an important source of control over the quality, defects evaluation and cost in software development. An effective measurement process requires continuous evaluation of different software metrics and integrating them into the software development process. Object-oriented metrics that can be used to measure the quality, Object-oriented design focus on measurements that are applied to the class and design characteristics.

In this paper, it is suggested that a combination of the methods should be used to Increase the Correctness and accuracy, how to improve the coding efficiency of an impact on the results when evaluating the project using object-oriented approaches based on model MOOD Metrics, CK metrics and COCOMO-II. Evaluating code using object-oriented approaches identifies some factors. This directly deals with the quality of the software. Using these results is very beneficial to improve software estimation, quality training, and research used for more accurate estimations of project milestones, and developing a software system that contains minimal faults.

Key Words: **Object oriented, Mood, CK, COCOMO, Defects, measurement, coding etc**

I. INTRODUCTION

Software engineering is the discipline which paves the roadmap for development of software's within given schedule and effort and with the desired quality. Object oriented design metrics is an essential part of software environment. The main objective of analyzing metrics is to improve the quality of the software.

Software measurements have become essential in software engineering. Many of the software developer's measure characteristics of the software to make sure those requirements are consistent and complete, the design is of high quality, and the code is ready to be tested. Effective project manager's measure attributes of process and product to be able to tell when the software should be ready for delivery and/or the budget has been exceeded.

Regular feedback from the development process is helpful in determining the status of the task and the project. Tracking gives opportunity to the project manager to take care of any unexpected situation.

II. PROJECT MANAGEMENT

Project Management is the organization and management of resource team in such a way that all the work required to complete a project can be done within defined scope, quality, and time and cost constraints. The purpose of project management is to first find out the activities needed to take the project to its end and secondly to allocate resources to these activities in a planned way.

III. SOFTWARE MEASUREMENT

Measurement programs in software organizations are an important source of control over quality, defects evaluation and cost in software development. Software measurement has evolved into a key software engineering discipline. Using some Object oriented methods identifies some key factor of coding and that are used for

IV. OBJECT ORIENTED DESIGN

Object-oriented design is a method of design encompassing the process of object oriented decomposing design is concerned with developing an object-oriented module of a software system to apply the identified requirements.

 DOI : 10.35291/2454-9150.2020.0091

Designer will use OOD because it is a faster development process, module based architecture, contains high reusable features, increases design quality and so on.

V. COCOMO II MODEL

Budgeting, planning and tracking, risk analysis and return on investments analyses are some of the uses of software cost, schedule and effort estimation. COCOMO II is one of the most widely used parametric models, for effort and schedule estimation.

VI. COMPARISON OF THE COCOMO II TOOLS

The tool developed as the result of thesis stores the information about the estimates and the actual data of the projects in to be used later by the tracking and calibration module and hence solves the problem of tracking and manual data feeding for calibration. measurement various model. *This directly deals with the quality of the software.*

Table 3.1: Comparison of the Tools

	CoStar	**Construx Estimate**	**COCOMO II 1999.0**	**SLIM-ESTIMATE**
Estimation	√	√	√	√
Estimation Model Used	COCOMO II	COCOMO II	COCOMO II	SLIM
Project Planning and Tracking	x	x	X	√
Report Generation	√	√	X	x
Calibration	√	√	√	√
Calibration Method Used	Regression Method	Regression Method	Regression Method	Regression Method

VII. STATEMENT OF THE PROBLEM

Study of tools has revealed the following drawbacks in the current scenario.

1. Not define Efficiency of developer and comparison between developers working ability.
2. During the development, the management needs to keep track of information about the status of project; the tools available do not have such features.
3. The method used for calibration of tools does not incorporate the expert's judgment in the resulting parameter values.
4. Tools available for the above activities are isolated to each other i.e. the tools available are either estimation tools or for planning and tracking also not compare between developers coding efficiency.
5. Any supporting documents or reports should be available to the person in the organization like SRS for the project, design specification. Current tools do not have this feature.
6. While calibration, past projects' data need to fetched manually.

VIII. INTERNAL QUALITY OF OOD

OO features	**Characteristics**	**Key Points**
Cohesion	Cohesion measures the degree of connectivity among the elements of a single class or object.	Cohesion can be used to identify the poorly designed classes.
Coupling	Coupling indicates the relationship or interdependency between modules	Coupling is a measure of interconnecting among modules in a software structure
Inheritance	Inheritance is the sharing of attributes and operations among classes based on a hierarchical relationship".	Reusability
Encapsulation	The process of compartmentalizing the elements of an abstraction that constitute its structure and behavior.	Hide the internal representation,or state, of an object from the outside.

IX. MOOD METRICS

F. B. Abreu proposed these system-level metrics. his set of six metrics measures four main structural mechanisms of object-oriented design that is encapsulation (Method Hiding Factor and Attribute Hiding Factor), inheritance (Method Inheritance Factor and Attribute Inheritance Factor), polymorphism (Polymorphism Factor) and message-passing (Coupling Factor).An explanation of the metrics with Java bindings follows except for coupling factor which was not measured

A system, based on the MOOD, has been developed to evaluate and grade Java programs. The interval of each MOOD metrics has been adapted, based on experimental results, to be fit in the evaluation of Java Programs. Also, a weight factor has been introduced to reflect the importance of each characteristic.

Software measurements have become increasingly essential in software engineering. Many of the best software developer's measure characteristics of the software to make sure that requirement are consistent and complete, the design is of high quality, and the code is ready to be tested.

The MOOD metrics are used to assess Java programs. The MOOD metrics consist of the following software quality indicators: Attribute Hiding Factor (AHF), Method Hiding Factor (MHF), Method Inheritance Factor (MIF), Attribute Inheritance Factor (AIF), Coupling Factor (COF), and Polymorphism Factor (POF).

 DOI : 10.35291/2454-9150.2020.0091

X. CK METRICS

1. Weighted Method per Class (WMC) – WMC measures the complexity of a class. WMC is a predictor of how much time and effort is required to develop and maintain the class. A large number of methods also mean a greater potential impact on derived classes, since the derived classes inherit the methods of the base class. If we analyze the WMC then we will find that high WMC leads to more faults which increases the density of bugs and decreases quality.

2. Depth of inheritance Tree (DIT) – DIT metric is the length of the maximum path from the node to the root of the tree. The deeper a class is in the hierarchy, the more methods and variables it is likely to inherit, making it more complex. High depth of the tree indicates greater design complexity. Thus it can be hard to understand a system with many inheritance layers. A high DIT has been found to increase faults and many methods might be reused.

3. Number of Children (NOC) – It is equal to the number of immediate child classes derived from a base class. NOC measures the breadth of a class hierarchy. A high NOC indicates several things like- High reuse of base class, base class may require more testing, improper abstraction of parent class etc.

4. Coupling Between Objects (CBO) - Two classes are coupled when methods declared in one class use methods or instance variables defined by the other classes. Multiple accesses to the same class are counted as one access. Only method calls and variable references are counted. An increase of CBO indicates the reusability of a class will decrease; also a high coupling has been found to indicate fault proneness. Thus, the CBO values for each class should be kept as low as possible.

5. Response for a Class (RFC) – The RFC is the count of the set of all methods that can be invoked in response to a message to an object of the class or by some method in the class.

This includes all methods accessible within the class hierarchy. Pressman, states that since RFC increases, the effort required for testing also increases because the test sequence grows. If RFC increases, the overall design complexity of the class increases and becomes hard to understand.

6. Lack of Cohesion (LCOM) - LCOM measures the dissimilarity of methods in a class by instance variables or attributes. A highly cohesive module should stand alone; high cohesion indicates good class subdivision. LCOM measures the amount of cohesiveness present, how well a system has been designed and how complex a class is. LCOM is account of the number of method pairs whose similarity is not zero. Lack of cohesion or low cohesion increases complexity, thereby increasing the likelihood of errors during the development process. If LCOM is high methods may be coupled to one another via attributes and then class design will be complex. So, designer should keep cohesion high, that is, keep LCOM low.

XI. FACTOR CALCULATION

1. Method Hiding Factor (MHF)

$$\text{MHF} = \frac{\sum_{i=1}^{TC}\sum_{m=1}^{Md(Ci)}(1 - V(Mmi))}{\sum_{i=1}^{TC} Md(Ci)}$$

Where:

$$V(Mmi) = \frac{\sum_{j=1}^{TC} is\_visible(Mmi,Cj)}{TC-1}$$

And:

$$is\_visible(Mmi, Cj) = \begin{cases} 1 & \text{iff } j \neq i \text{ and Cj may call Mmi} \\ 0 & \text{otherwise} \end{cases}$$

MOOD-Java Binding:
TC– total number of classes in the system/package.
Md(Ci) – number of constructors and methods defined with any access modifier excluding abstract and inherited methods.

2. Attribute Hiding Factor (AHF)

$$\text{AHF} = \frac{\sum_{i=1}^{TC}\sum_{m=1}^{Ad(Ci)}(1 - V(Ami))}{\sum_{i=1}^{TC} Ad(Ci)}$$

Where:

$$V(Mmi) = \frac{\sum_{j=1}^{TC} is\_visible(Ami,Cj)}{TC-1}$$

And:

$$is\_visible(Ami, Cj) = \begin{cases} 1 & \text{iff } j \neq i \text{ and Cj may reference Ami} \\ 0 & \text{otherwise} \end{cases}$$

MOOD-Java Binding: Ad (Ci) – number of all attributes with any access modifier but not including inherited.

3. Method Inheritance Factor (MIF)

$$\text{MHF} = \frac{\sum_{i=1}^{TC} Mi(Ci)}{\sum_{i=1}^{TC} Ma(Ci)}$$

Where Ma (Ci) = Md (Ci) + Mi (Ci)

The numerator is the sum of inherited methods in all classes of the system. The denominator's the total number of available methods in all classes.

MOOD-Java Binding:

Mi (Ci) – number of inherited methods but not overridden

Md (Ci) – number of defined non-abstract methods with any access modifier.

Ma (Ci) – number of methods that class Ci can call.

4. Attribute Inheritance Factor (AIF)

$$\text{AIF} = \frac{\sum_{i=1}^{TC} Ai(Ci)}{\sum_{i=1}^{TC} Aa(Ci)}$$

 DOI : 10.35291/2454-9150.2020.0091

Where, Aa (Ci) = Ad (Ci)+Ai (Ci)

It is defined analogous to MIF.

MOOD- Java Binding:

Ai (Ci) – number of inherited attributed

Ad(Ci) – number of defined attributes with any access Modifier.

Aa(Ci) – number of attributes that Class Ci can reference.

XII. PROGRAMMER CODING EVALUATION AND PERFORMANCE MEASUREMENT

Our approach focused programmer efficiency and working capability. the concept software metrics evaluation and analysis application and provides metrics results by applying Chidamber & Kemerer and MOOD metrics to several standard Java libraries and projects. It also introduces Java bindings for these set of metrics. The analysis of the results reveals helpful cognitions about how object-oriented approach is being implemented by these technologies. In addition to providing common trends in metrics values, this information can be combined with validation studies from other researchers to adapt software design to proven approaches and hence avoid possible expensive maintenance tasks and optimize design for better quality software The tool was developed with the intention to be used by project management team which bears the responsibility of completing the project within time, budget and with the specified quality.

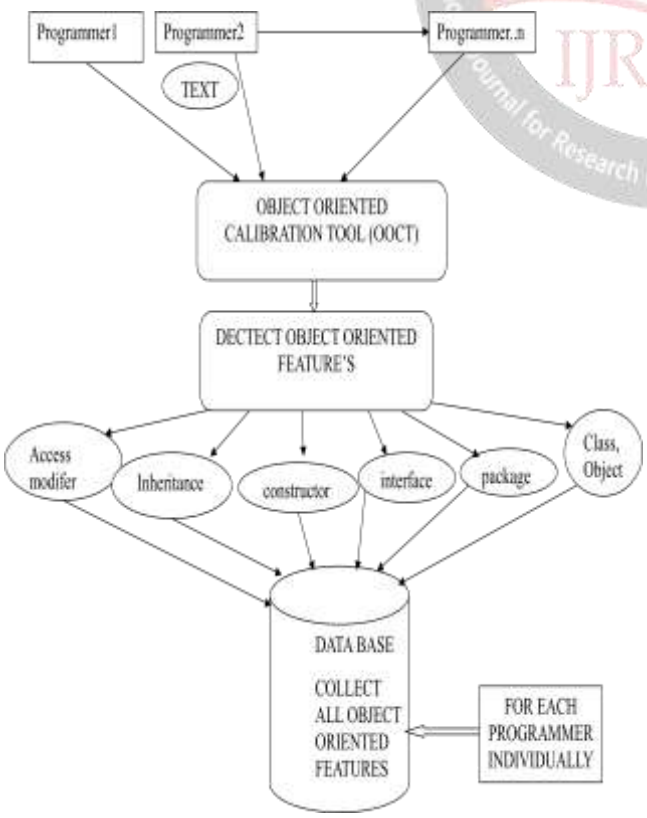

Fig1. Programmer coding key evaluation

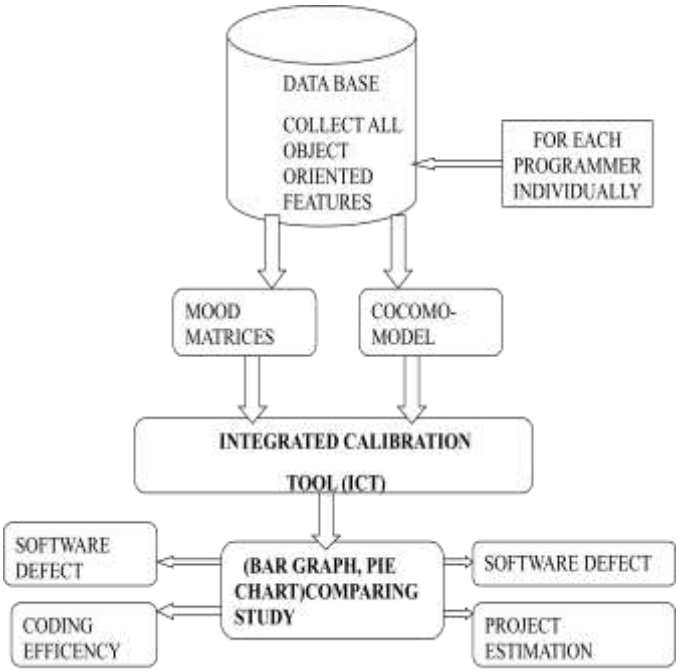

Fig2. Coding evaluation & performance measurement

A metrics based analysis of various programming language libraries can expose structural and design commonalities among them. Thus we can obtain more generalized view of software design heuristics. The analysis data can also reveal comparative inherent complexity within various standard libraries which is likely to be inherited to the software applications using those libraries. Some other Future Research Direction.

XIII. DEFECT EVALUATION

The presence of coding defects in object oriented code can have a severe impact on the quality of software. The detection and correction of coding defects is an important issue for cost effective maintenance. They increase the performance of the software. we propose an automatic coding evaluation technique which uses the comparison past and current code are reference for good coding to detect the defects in existing software coding. We also propose the correction technique which can refactor the code to meet the coding specifications to improve coding efficiency.

XIV. FUTURE WORK

Analyzes a specific set of object-oriented metrics for various Java technology libraries. A similar analysis can be performed for other competing technologies such as .NET C++ etc.

CK and MOOD belong to the class of structural and complexity metrics.

- Programmer coding evaluation using multimedia data.
- Training and Research area.
- Reduce execution time and space complexity.

 DOI : 10.35291/2454-9150.2020.0091

- Better report generation of the project which can be a blueprint for forwarding engineering process
- Better HR management.

REFERENCES

[1] 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.

[2] Ping Cao a , Ke Yang b, Ke Liu c "Optimal selection and release problem in software testing process: A continuous time stochastic control approach" European Journal of Operational Research March 2, 2019.

[3] Magne Jørgensen and Martin Shepperd, "A Systematic Review of Software evelopment Cost Estimation Studies," IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 33, NO. 1, JANUARY 2007.

[4] Tirimula Rao Benalaa, Rajib Mallb "DABE: Differential evolution in analogy-based software development effort estimation"Swarm and Evolutionary Computation 38 (2018) 158–172.

[5] 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.

[6] Barbara A. Kitchenham,Robert T. Hughes, and Stephen G. Linkman, "Modeling Software Measurement Data," IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 27, NO. 9, SEPTEMBER 2001.

[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.

 DOI : 10.35291/2454-9150.2020.0091