Parul, 2021, 9:4
ISSN (Online): 2348-4098
ISSN (Print): 2395-4752

Fault Prediction using Object Oriented Metrics: CK Metrics and MOOD Metrics

Research Scholar Parul
Department of Computer Science,
Baba Mastnath University,
Rohtak, India.

Asst. Prof. Deepak Kumar, Asst. Prof. Manjeet Kaur
Department of Computer Science,
Govt. College Sector 1,
Panchkula, India

Abstract- It is widely accepted that object-oriented development requires a different way of thinking than traditional structured development and software projects are shifting to object-oriented design. In structured approach, the problem is divided into functions. Each function has its own data and logic. This structured approach has following limitations. Data is given second importance where as function is given first importance. But it is known that data is most important than function. The existence of function is due to data. This approach does not model the real world very well. These above limitations are overcome by object oriented approach. In object-oriented approach, problem is divided into objects. Objects contain the data and function that operate on the data. Data and its functions are encapsulated in to a single entity i.e. object. This approach represents the real world very well and data is given more importance in comparison to function. The main advantage of object-oriented design is its modularity and reusability. Object-oriented metrics are used to measure properties of object-oriented designs. Metrics are a means for attaining more accurate estimations of project milestones, and developing a software system that contains minimal faults. Project based metrics keep track of project maintenance, budgeting etc. Design based metrics describe the complexity, size and robustness of object-oriented and keep track of design performance. Compared to structural development, object-oriented design is a comparatively new technology. The metrics, which were useful for evaluating structural development, may perhaps not affect the design using OO language. As for example, the "Lines of Code" metric is used in structural development whereas it is not so much used in object-oriented design. One study estimated corrective maintenance cost saving of 42% by using object-oriented metrics [21].

Keywords:- Object-oriented designs, project based metrics, structural development etc.

I. INTRODUCTION

It is widely accepted that object-oriented development requires a different way of thinking than traditional structured development and Software projects are shifting to object-oriented design. In structured approach, the problem is divided into functions. Each function has its own data and logic. This structured approach has following limitations. Data is given second importance where as function is given first importance. But it is known that data is most important than function. The existence of function is due to data. This approach

International Journal of Science, Engineering and Technology

An Open Access Journal

does not model the real world very well. These above limitations are overcome by object oriented approach. In object-oriented approach, problem is divided into objects. Objects contain the data and function that operate on the data. Data and its functions are encapsulated in to a single entity i.e. object. This approach represents the real world very well and data is given more importance in comparison to function.

The main advantage of object-oriented design is its modularity and reusability. Object-oriented metrics are used to measure properties of object-oriented designs. Metrics are a means for attaining more accurate estimations of project milestones, and developing a software system that contains minimal faults. Project based metrics keep track of project maintenance, budgeting etc. Design based metrics describe the complexity, size and robustness of object-oriented and keep track of design performance. Compared to structural development, object-oriented design is a comparatively new technology.

The metrics, which were useful for evaluating structural development, may perhaps not affect the design using OO language. As for example, the "Lines of Code" metric is used in structural development whereas it is not so much used in object-oriented design. One study estimated corrective maintenance cost saving of 42% by using object-oriented metrics [21].

II. REVIEW OF LITERATURE

Chidamber and Kemerer [9] proposed a suite of object-oriented design metrics which were developed based on the ontology of Bunge. They analytically evaluated the metrics against Weyuker's measurement theory principles [26] and provided an empirical sample of these metrics from two commercial systems. Several studies have been conducted to validate CK metrics. Basili, Briand and Melo [4] presented the results of an empirical validation of CK metrics. Based on eight medium sized school projects they applied a logistic regression model to investigate whether these metrics can be used as fault-prone class indicators. Their results suggest that five of the six CK metrics are useful quality indicators for predicting fault-prone classes.

Li and Henry [17] used two size metrics and eight OO metrics, including five of CK metrics, to empirically validate the applicability of these metrics on the number of lines changed per class, assumed to be related to maintenance effort. This empirical validation was conducted on two commercial systems using multiple linear regression technique. Their results show that OO metrics can be used to predict maintenance effort, measured by the number of lines changed per class, in an object-oriented system. Li [16] also theoretically validated CK metrics using metric-evaluation framework proposed by Kitchenham et. al. [15].

He discovered some deficiencies of CK metrics in the evaluation process and proposed a new suite of OO metrics that overcome these deficiencies. Chidamber et. al. [8] further explored the applicability of CK metrics on practical managerial work such as productivity and rework effort. Their empirical results suggest CK metrics were significant economic variable indicators for the three commercial OO systems used in their study. Analyzing a medium-sized telecommunication system, Cartwright and Shepperd [7] studied the inheritance measures from the CK suite (DIT, NOC) and found that both these measures were associated with defect density of classes.

Selvarani, Nair & Prasad [22] building a quality system has been the driving goal of all software engineering efforts over few decades. The lack of design and implementation guidance may affect the overall quality of the system which depends on reusability, defect level and maintainability of the system. A significant research effort is required to define quality measures. Measuring the structural design properties of software artifacts with design metrics, is a promising approach at an early stage.

Our estimation model provides an assessment of the defect proneness of the system in an early stage by analyzing the interrelationship among the defect occurrence and design parameters of the software.
Subramanyan & Krishnan [24] study enhances prior empirical literature on OO metrics by providing a new set of results validating the association between a subset of CK metrics and defects detected during acceptance testing and those reported by customers. One of our main findings is that, after controlling for size, we find that some of the

International Journal of Science, Engineering and Technology

An Open Access Journal

measures in the CK suite of OO design complexity metrics significantly explain variance in defects.

Tang, Kao, Chen [25] validated CK metrics using three industrial real-time systems and the results suggest that WMC can be a good indicator for faulty classes and RFC is a good indicator for OO faults. Furthermore, presented a set of new metrics which considered useful as indicators of OO fault-prone classes. Therefore, these new metrics can be utilized to decide which classes need to be tested using OO testing techniques.

III. OBJECTIVE OF THE RESEARCH

- To study various object-oriented metrics.
- To study fault prediction using various object-oriented metrics.
- To investigate the impact of faults on object oriented software to improve the quality.

IV. RESEARCH METHODOLOGY

Object-oriented design has become a dominant method in software industry and many design metrics of object oriented programs have been proposed for quality prediction, but there is no well-accepted statement on how significant those metrics a The object oriented metrics will be adopted to identify a limited set of measureable attributes that have a significant impact on prediction of Faults and quality attributes. The techniques involved will be statistical analysis.The statistical techniques will be used to reveal the relationship between metrics and dependent variables.

1. The Proposed Plan of Work:

The starting of the dissertation would be devoted on the introduction of various existing object-oriented design metrics. The second step would focus on study of software defects using complexity metrics in object-oriented design. The third step would focus on study of faults in object-oriented design. The fourth step would focus on impact faults on the quality of object-oriented software. Lastly the summary and conclusions and scope for further research would be discussed.

Object-oriented design has many useful qualities, such as cohesion, coupling, inheritance, encapsulation, information hiding, localization etc.

- **Cohesion** refers to the internal consistency within the parts of the design. Cohesion is centred on data that is encapsulated within an object and on how methods interact with data to provide well-bounded behaviour. A class is cohesive when its parts are highly correlated. It should be difficult to split a cohesive class. Cohesion can be used to identify the poorly designed classes. "Cohesion measures the degree of connectivity among the elements of a single class or object"[5].

- **Coupling** indicates the relationship or interdependency between modules. For example, object X is coupled to object Y if and only if X sends a message to Y that means the number of collaboration between classes or the number of messages passed between objects. Coupling is a measure of interconnecting among modules in a software structure.

- **Inheritance** is a mechanism whereby one object acquires characteristics from one, or more other objects. Inheritance occurs in all levels of a class hierarchy. "Inheritance is the sharing of attributes and operations among classes based on a hierarchical relationship" [20].In general, conventional software does not support this characteristic because it is a pivotal characteristic in many object-oriented systems as well as many object-oriented metrics focus on it.

- **Encapsulation** is a mechanism to realize data abstraction and information hiding. Encapsulation hides internal specification of an object and show only external interface. "The process of compartmentalizing the elements of an abstraction that constitute its structure and behaviour; encapsulation serves to separate the contractual interface of an abstraction and its implementation"[5].Encapsulation influences metrics by changing the focus of measurement from a single module to a package of data.

- **Information Hiding** is the process of hiding all the secrets of an object that do not contribute to its essential characteristics. An object has a public interface and a private representation; these two elements are kept distinct. Information hiding acts a direct role in such metrics as object coupling and the degree of information hiding.

International Journal of Science, Engineering and Technology

An Open Access Journal

"All information about a module should be private to the module unless it is specifically declared public"[1].

- **Localization** is based on objects in object-oriented design approach. In a design, if there are some changes in the localization approach, the total plan will be violated, because one function may involve several objects, and one object may provide many functions.

  "Localization is the process of gathering and placing things in close physical proximity to each other"[3].

Metrics should apply to the class as a complete entity. Even the relationship between functions and classes is not necessarily one-to-one. For that reason, metrics that reflect the manner in which classes collaborate must be capable of accommodating one-to-many and many-to-one relationships [19].

In the object-oriented environment, one of the major aspects having strong influence on the quality of resulting software system is the design complexity.

The structural property of the software component is influenced by the cognitive complexity of the individuals involved in designing, development and testing, and it will be reflected in the structural properties of the developed software. This cognitive complexity is likely to affect other aspects of these components, such as fault-proneness and maintainability.

The OO paradigm offers the technology to create components that can be used for generic programming [22]. Design complexity has been conjectured to play a strong role in the quality of the resulting software system in OO development environments [5]. Design complexity in traditional development methods involved the modeling of information flow in the application.

Hence, graph-theoretic measures [18] and information-content driven measures [14] were used for representing design complexity. In the OO environment, certain integral design concepts such as inheritance, coupling, and cohesion have been argued to significantly affect complexity. Hence, OO design complexity measures proposed in literature have captured these design concepts [24].

One of the first suites of OO design measures was proposed by Chidamber and Kemerer (CK) [10], [9]. The authors of this suite of metrics claim that these measures can aid users in understanding design complexity, in detecting design flaws and in predicting certain project outcomes and external software qualities such as software defects, testing, and maintenance effort. Use of the CK set of metrics and other complementary measures are gradually growing in industry acceptance [24].

CK metrics suite [9] is one of the object-oriented design complexity measurement systems which support the measurement of the external quality parameter which may evolve in software package. The literature widely refers to the metric suite which depends on the internal structural analysis of object-oriented components such as inheritance, coupling, cohesion, method invocation, and association [22].

2. CK Metrics:

The Chidamber and Kemerer have proposed six class-based design metrics for object-oriented systems [23][6].

- **Coupling Between Objects (CBO).** The CBO metric counts the number of other classes to which a class is coupled. It counts the number of reference types that are used in attribute declarations, formal parameters, return types, throws declarations, local variables, and types from which attribute and method selections are made. Primitive types, types from the java.lang package, and supertypes are not counted. High values of CBO metrics mean that the class is highly coupled. The developers and testers perceive that the maintainability and testability of highly coupled classes is difficult.

  which makes the process of maintaining and uncovering faults prerelease and postrelease difficult as well. The viewpoint are: If small values of CBO then improve modularity and promote encapsulation, indicates independence in the class making easier its reuse, makes easier to maintain and to test a class.

- **Lack of Cohesion of Methods (LCOM).** The LCOM metric is the number of pairs of methods in the class using no attributes in common (referred to as P), minus the number of pairs of methods that do (referred to as Q). The LCOM is set to zero if this

International Journal of Science, Engineering and Technology

An Open Access Journal

difference is negative. After considering each pair of methods: LCOM = (P>Q) ? (P-Q) : 0. The LCOM metric measures the coherence among local methods in a class. The class that does one thing (i.e., cohesive class) is easier to reuse and maintain than the class that does many different things (i.e., the class provides many different services). The viewpoints are: If great values of LCOM then increases complexity, does not promotes encapsulation and implies classes should probably be split into two or more subclasses and helps to identified low-quality design.

- **Weighted Methods Complexity (WMC).** The WMC metric is the sum of the complexity of all methods for a class. Normally, many metrics tools calculate the WMC metric as simply the number of methods in a class. This is equivalent to saying all functions have equal complexity. However, in this research, the tool we used (i.e., Borland Together) calculates the WMC metric by summing the McCabe cyclomatic complexity of all the methods in the class.

Therefore, high values of the WMC metric mean high complexities as well. The viewpoints are: WMC is a predictor of how much time and effort is required to develop and to maintain the class, the larger Number of Method (NOM) the greater the impact on children. Classes with large NOM are likely to be more application specific, limiting the possibility of reuse and making the effort expended one shot investment.

- **Depth of Inheritance Hierarchy (DIT).** The DIT measures the length of the inheritance chain from the root of the inheritance tree to the measured class. The DIT metric is an indicator of the number of ancestors of a class. It may require developers and testers to understand all ancestors to comprehend all specializations of the class, which is necessary to maintain or uncover pre and post release faults.

The viewpoints are: If the greater values of DIT then the greater the Number of Methods (NOM) it is likely to inherit, making more complex to predict its behavior, the greater the potential reuse of inherited methods. Small values of DIT in most of the system's classes may be an indicator that designers are forsaking reusability for simplicity of understanding.

Number of Child Classes (NOC). The NOC metric counts the number of descendents of a class. The number of children represents the number of specializations and uses of a class. Therefore, understanding all children classes is important to understand the parent. The high number of children increases the burden on developers and testers in comprehending, maintaining, and uncovering pre and post release faults.

The viewpoints are: If the greater is the NOC then the greater is the reuse, the greater is the probability of improper abstraction of the parent class, the greater the requirements of methods testing in that class. Small values of NOC, may be and indicator of lack of communication between different class designers.

Response for class (RFC). The size of the response set for the class includes methods in the class's inheritance hierarchy and methods that can be invoked on other objects. The RFC metric counts the number of methods in the response set for a class, which includes the number of local methods and the number of remote methods invoked by local methods. The class that has a large number of responsibilities tends to be large and has many interactions with other classes. Therefore, such classes are complex and incur more time and effort to maintain and test than small classes. The viewpoints are: If a large numbers of methods are invoked from a class (RFC is high) then testing and maintenance of the class become more complex.

3. MOOD Metrics:

Abreu et at. defined MOOD (Metrics for Object Oriented Design) metrics[24,25,26]. MOOD refers to a basic structural mechanism of the object-oriented paradigm as encapsulation (MHF, AHF) inheritance (MIF, AIF), polymorphism (POF), and message passing (COF). We will discuss MOOD metrics in the context of encapsulation, inheritance, polymorphism, and coupling. These are discussed below:

3.1 Encapsulation: The Method Hiding Factor (MHF) and Attribute Hiding Factor (AHF) were proposed together as measure of encapsulation

Method Hiding Factor (MHF). This metric is the ratio of hidden (private or protected) methods to total methods. As such, MHF is proposed as a measure of encapsulation. If the value of MHF is high (100%), it means all methods are private which indicates very little functionality. Thus it is not possible to reuse methods with high MHF. MHF with low (0%) value

indicate all methods are public that means most of the methods are unprotected.

Attribute Hiding Factor (AHF). This metric is the ratio of hidden (private or protected) attributes to total attributes. AHF is also proposed as a measure of encapsulation. If the value of AHF is high (100%), it means all attributes are private. AHF with low (0%) value indicates all attributes are public.

3.2 Inheritance: Inherited features in a class are those which are inherited and not overridden in that class. Method Inheritance Factor (MIF) and Attribute Inheritance Factor (AIF) are proposed to measure inheritance.

Method Inheritance Factor (MIF). This metric is a count of the number of inherited methods as a ratio of total methods. If the value of MIF is low (0%), it means that there is no methods exists in the class as well as the class lacking an inheritance statement

Attribute Inheritance Factor (AIF). This metric counts the number of inherited attributes as a ratio of total attributes. If the value of AIF is low (0%), it means that there is no attribute exists in the class as well as the class lacking an inheritance statement.

3.3 Polymorphism: Polymorphism is an important characteristic in object oriented paradigm. Polymorphism measure the degree of overriding in the class inheritance tree.

Polymorphism Factor (PF). This metric is based on the number of overriding methods in a class as a ratio of the total possible number of overridden methods. The value of POF can be varies between 0% and 100%. If a project have 0% POF, it indicates the project uses no polymorphism and 100% POF indicates that all methods are overridden in all derived classes

3.4 Coupling:
Coupling shows the relationship between modules. A class is coupled to another class if it calls methods of another class.

Coupling Factor (CF). This metric counts the number of inter-class communications. The value of COF can be varies between 0% and 100%. 0%COF indicates no class are coupled and 100% COF indicates all class are coupled with all other classes. High values of COF should be avoided.

REFERENCES

[1] Balasubramanian NV.: "Object-oriented metrics", Proceedings 3rd Asia-Pacific Software Engineering Conference (APSEC'96). IEEE Computer Society, 1996; 30-34.
[2] Banker R. D., Datar S. M., Kemerer C.F., and Zweig D., "Software Complexity and Software Maintenance Costs," Comm. ACM, vol. 36, no. 11, pp. 81-94, 1993.
[3] Banker, Rajiv D., Kauffman, Robert J.,Kumar, Rachina.: "An Empirical Test of Object-based Output Measurement Metrics in a CASE Environment." Journal of Management Information Systems 8,3 (Winter 1991): 127-150.
[4] Basili V.R., Briand L.C., and Melo W.L., "A validation of object-oriented design metrics as quality indicators" IEEE Transactions on Software Engineering, 22(10):751–761, October 1996.
[5] Booch G., "Object-Oriented Analysis and Design with Applications", second ed. Redwood City, Calif.: Benjamin/Cummings, 1994.
[6] Camargo Cruz Ana Erika, " Chidamber & Kemrer Suite of Metrics", Japan Advanced Institute of Science and Technology School of Information, May 2008.
[7] Cartwright M. and Shepperd M., "An Empirical Investigation of an Object-Oriented Software System," IEEE Trans. Software Eng., vol. 26, no. 7, pp. 786-796, Aug. 2000.
[8] Chidamber S.R., Darcy D.P., and Kemerer C.F., Managerial use of metrics for object-oriented software: An exploratory analysis. IEEE Transactions on Software Engineering, 24(8):629–639,August 1998.
[9] Chidamber S. and Kemerer C.: "A Metrics Suite for Object-oriented Design", IEEE Transactions on Software Engineering, vol. 20, no. 6, pp. 476-493, June 1994.
[10] Chidamber S. R., and Kemerer C.F., "Towards a Metrics Suite for Object-oriented Design," Proc. Conf. Object-oriented Programming Systems, Languages, and Applications (OOPSLA'91), vol. 26, no. 11, pp. 197-211, 1991.
[11] Emam K. EL., Benlarbi S., Goel N., and Rai S. N., "The Confounding Effect of Class Size on the Validity of Object-Oriented Metrics," IEEE Trans. Software Eng., vol. 27, pp. 630-650, 2001.

International Journal of Science, Engineering and Technology

An Open Access Journal

[12] Fenton N. E., "Software Metrics: A Rigorous Approach" London: Chapman and Hall, 1991.
[13] Grady R. B. and Caswell D. L., Software Metrics: Establishing a Company-Wide Program. New Jersey: Prentice Hall, 1987.
[14] Halstead M., Elements of Software Science. New York: Elsevier North-Holland, 1977.
[15] Kitchenham B., Pfleeger S.L., and Fenton N., "Towards a framework for software measurement validation" IEEE Transactions on Software Engineering, 21(12):929–944, December 1995.
[16] Li W., "Another metric suite for object-oriented programming". Journal of Systems and Software, 44:155–162, 1998.
[17] Li W. and Henry S., "Object-oriented metrics that predict maintainability" Journal of Systems and Software, 23:111–122, 1993.
[18] McCabe T.J., "A Complexity Measure," IEEE Trans. Software Eng.,vol. 2, pp. 308-320, 1976
[19] Rosenberg, H Linda: "Applying and Interpreting Object-oriented Metrics" Software Assurance Technology Office (SATO).
[20] Rumbaugh, J.,Blaha, M., Premerlani,W., Eddy F. And Lorenses, W: Object-oriented modeling and design, Prentice Hall, 1991.
[21] Sarker M., "An overview of object-oriented design metrics", Thesis, Umea University, Sweden, pp.9-10, June 2005
[22] Selvarani R., Nair T.R.G., Prasad V.K., "Estimation of defect proneness using design complexity measurements in object-oriented software", IEEE computer society: pp 766-770, 2009.
[23] Shatnawi R., " A quantitative investigation of the acceptable risk levels of object-oriented metrics in open-source systems", IEEE Transactions on Software Engineering, Vol. 36, No.2, pp. 223-224 March/April 2010.
[24] Abreu F.B. and R.Carapuca."Object-Oriented Software Engineering: "Measuring and Controlling the Development Process". Proceedings of the 4th International Conference on Software Quality, McLean,Virginia, USA,October ,1994.
[25] Abreu, B.F. and W.L. Melo (1996): "Evaluating the impact of Object-Oriented Design on Software Quality", Proceedings of METRICS '96, IEEE, 1996.pp. 90-99
[26] Abreu F.B.(1995):. ECOOP'95 Quantitive Methods Workshop"Design Metrics for Object-Oriented Software Systems"1995
[27] Subramanyam R., Krishnan M.S., "Empirical analysis of CK metrics for object-oriented design complexity: implications for software defects Software Engineering", IEEE Transactions on Publication Date: April 2003 Volume: 29, Issue: 4 On page(s): 297- 310.
[28] Tang M. H., Kao M. H., Chen M. H., "An empirical study on object-oriented metrics", Computer Science Department SUNY at Albany, NY 1222.
[29] Weyuker E., "Evaluating software complexity measures" IEEE Transactions on Software Engineering, 14:1357–1365, 1988.