# An Overview of Quality Metrics Used in Estimating Software Faults

Talha Burak Alakus
*Department of Software Engineering*
*Technology Faculty, Firat University*
Elazig, Turkey
burak.alakuss@gmail.com

Resul Das
*Department of Software Engineering*
*Technology Faculty, Firat University*
Elazig, Turkey
rdas@firat.edu.tr

Ibrahim Turkoglu
*Department of Software Engineering*
*Technology Faculty, Firat University*
Elazig, Turkey
iturkoglu@firat.edu.tr

***Abstract*—Software reliability is highly affected by software quality attributes and measurements. Faults, bugs, and errors are shown not only in the development process but also in end-user period hereby it is required to detect these issues earlier. These are detected by software quality and object-oriented metrics which are commonly used in the fault detection process. CK, MOOD and QMOOD metrics are the most common metrics applied in this area. In this paper is to aim to provide information about popular software quality metrics and their usage in terms of software fault prediction studies. For this purpose, in this work, these three metrics were analyzed separately and their acquisition methods were showed. Moreover, in order for the software to survive with errors, to remove and minimize errors, a number of recommendations are presented in the conclusion section.**

***Index Terms*—Software faults, software metrics, software engineering, quality attributes.**

## I. Introduction

Development organizations and industries endeavor the reach the high software quality in earlier stage as possible. In accordance with the developers, the quality of a good software is related to three things: *i)* content of errors, faults, defects or bugs, *ii)* intensity of these faults, and *iii)* fault proneness which all are represents the quality of software metrics [1]. Software reliability and quality can be measured with different techniques including testing and maintenance, software fault proneness and fault prediction [2]. Testing of a software nearly consumes of all resources of the projects and approximately between 50% and 60% of them are used to test the software [3]. In a software project, almost 20% of the software modules generate faults of 80% [4]. Thus, predicting the software faults before testing process is time and cost efficient. Software fault prediction studies aim to find defects, bugs, errors and faults during software development process [5] and to improve the quality and reliability of the software. Software and object-oriented metrics are required for software fault prediction studies and there are many metrics have been proposed in this area. CK (Chidamber and Kemerer) metrics [6], [7] MOOD (Abreu and Carapuca) metrics [8] and QMOOD (Bansiya and Davis) metrics [9] are the most important examples of these metrics. Software metrics are used to determine the quality of the software based on analyzing different aspects of software complexity. They provide valuable information such as its complexity, reusability, reliability, maintainability which are the external quality attributes of the software and design purposes [24]. Software metrics can be divided into two categories: Product and process. The information for each type of metrics has given below [14];

- Product Metrics: They are generally used to evaluate the attributes of the software. Product metrics also known quality metrics composed of different metrics and properties including complexity, size, style, usability, functionality, performance and non-reliability.
- Process Metrics: They are effective to measure the attributes of the software process in a software product. Process metrics also known management metrics composed of different metrics and properties including cost, advancement, effort and reusability.

They have a good impact on software by helping and improving the quality of a system and designate the existing system size with respect schedule and cost by comparing the existing software products. Like any other things, metrics also have both advantages and disadvantages. Advantages can be emphasized as follows:

- It is helpful for determining the complexity of a system.
- It provides the testing resources early development of project life cycle.
- It is useful to notify the project owners the current status of the software.
- It provides the necessary analysis to comprehend the design and development process of a system.
- Beneficial to validate and verify the software based on the functional and nonfunctional requirements and their specifications.

Disadvantages of metrics can be specified as follows:

- Theoretically metrics are easy to implement yet in some cases it is not easy as expected. In some software development events, it requires a specific budget and more complex.
- Many models of the software development process are required experimental and stochastic operations. Thus, it is difficult to implement the design metrics in such cases.
- The performance of software metrics based on the development tools, working environment and application

programs. In some cases, it suffers from the objectivity.
- Software metrics only analyze the attributes of a system. What about technical and administrative staff?
- In software development models, different attributes including requirements analysis, cost, size of project, success rate, flexibility, risk management, simplicity, source management, and cost management effect the models diversely in different phases [10], [11]. Thus, object-oriented metrics provide different information on each development phases.

These metrics do not improve the quality of their skills. Object oriented metrics were defined to measure the software attributes and provide the relationships between the software complexity metrics. Complexity effects the understandability, replaceability, testability and faults of the software which are general aspects of the software maintainability. Figure 1 depicts the general information of metrics, their relationships and effects on the software faults.

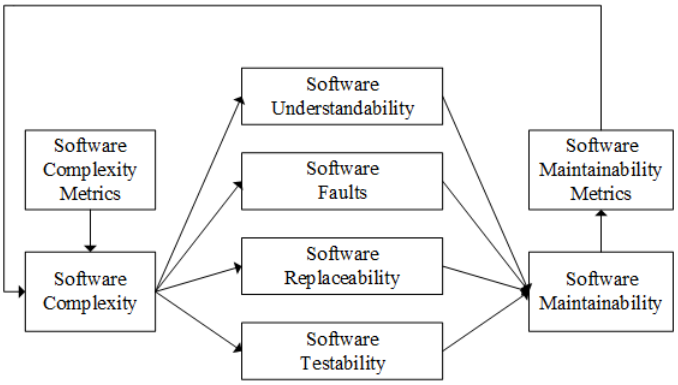


Fig. 1. Relationship of software metrics in general.

The purpose of this study is to provide the definition of these metrics including their collection process. Each of these metrics analyzed separately and we showed what is the meaning of these metrics, their values and how to obtain them. The remainder of this paper is organized as follows: In Section 2, information about each metrics has given. Definitions of the metrics values also have been provided. In Section 3, the collection of these values has been shown. In the last section study is concluded.

## II. Software and object-oriented metrics for software fault prediction

In this study three different metrics were analyzed: CK, MODD, and QMOOD.

### A. Chidamber and Kemerer metrics suite

The study of Chidamber and Kemerer were originally proposed in 1991 yet their study has been criticized cruelly based on lack of theoretical information, lack of attributes of measurement methods, and labor-intensive [12]–[15]. Therefore, metrics were analyzed in 1994 with regards to develop and validate these metrics of object-oriented design by providing theoretical and practical information. They developed six design metrics. These design metrics and their background information has been given below in detail:

- WMC (Weighted Methods per Class): WMC can be described as the number of defined methods in a class. It is used to measure the complexity, re-usability, understandability and maintainability of a software. The high number of WMC refers to the software is more complex and the less understandable, reliable, and reusable [16].
- DIT (Depth of Inheritance): DIT calculates the depth of a class. Generally used to measure maintainability and re-usability of software. The low number of DIT refers to the software is tend to be more reusable [17]. Yet, increasing the DIT value makes the system more difficult to maintain and reuse.
- NOC (Number of Children): NOC calculates the number of classes related to the given class which has an inheritance relationship. It measures the complexity and maintainability of the software. Higher NOC value means bad design and it makes the system more complex and less maintainable [17].
- CBO (Coupling Between Objects): CBO calculates the number of coupled classes. It is used to designate whether a class is using a variable in other classes or not. It measures the complexity, maintainability, and re-usability of the software [7], [18]. High number CBO makes the system less reusable and less maintainable yet increases the complexity.
- RFC Response for a Class: RFC is the number of methods defined in a class and called from a class. It is used to measures the complexity, maintainability, and quality of the software. The high number of RFC causes the system is more complex and as a result of that maintainability and quality decreases [17].
- LCOM (Lack of Cohesion of Methods): LCOM calculates the difference between the methods. The equation of calculation has given in Equation 1 [7]:

$$LCOM = |P| - |Q| \quad if \quad |P| > |Q|, 0 \quad otherwise \quad (1)$$

P represents the number of dissimilar methods while Q means the number of similar methods. The similarity is determined by the number of attributes which is used commonly for both methods. It affects the complexity of the system and low LCOM makes the system more complex [18].

Effect on each Chidamber and Kemerer metrics on various software quality attributes can be seen in Figure 3.

### B. Abreu and Carapuca metrics suite

In their study, they proposed metrics for object-oriented design approaches including inheritance, information hiding, encapsulation, and polymorphism. By developing these matrices, they aimed to enhance the quality of software and development productivity. Metrics were developed based on 7 various criterions which are mentioned below [8]:

1) Metrics should be defined objectively.

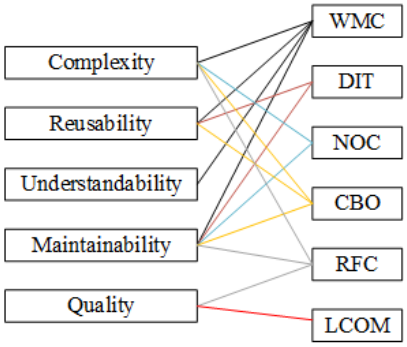


Fig. 2. Relationship of C&K metrics with software quality attributes.

2) Metrics size should be independent of system-size.
3) Metrics should not have dimension.
4) Metrics should be available in the early stages of life-cycle for collecting.
5) Metrics should be scalable.
6) Metrics should be computable straightforwardly.
7) Metrics should be understandable in every language.

6 different design metrics were proposed based on these criterions and named as MOOD (Metrics for Object-Oriented Design). In below, their background information and description has been given:

1) MHF (Method Hiding Factor): MHF measures functionality of the class. A low MHF means the functionality of the system is high. On the other hand, when this value increases, the system functionality lowers [19], [20].
2) AHF (Attribute Hiding Factor): AHF measures the invisibilities of the variables, data members, and fields in a given class [21].
3) MIF (Method Inheritance Factor): MIF is a part of inheritance measurement and evaluates the system reusability [22]. MIF value is 0 when class does not have any methods or when there is no inheritance hierarchy.
4) AIF (Attribute Inheritance Factor): AIF like MIF is a part of inheritance measurement of a system. It is generally used to determine whether the system is reusable or not [22]. AIF value is 0 when class does not have any attributes (variables, fields, etc.) or when there is no inheritance hierarchy.
5) CF (Coupling Factor): CF or COF is used to measure the communication between client class and supplier class. It is ideal when a given class established a lower communication with other classes or they share little information to each other [23].
6) PF (Polymorphism Factor): PF measures the polymorphism of a class. It is used to evaluate the degree of dominant methods in an inheritance tree [24].

### C. Bansiya and Davis metrics suite

Authors developed an object-oriented design metrics to enhance the both behavioral and structural properties of objects, classes and their relationships. Metrics were defined based on the design properties including coupling, encapsulation, cohesion, and modularity to determine the software flexibility, complexity, maintainability, re-usability, etc. 11 different design metrics were proposed and named as QMOOD (Quality Model for Object-Oriented Design).

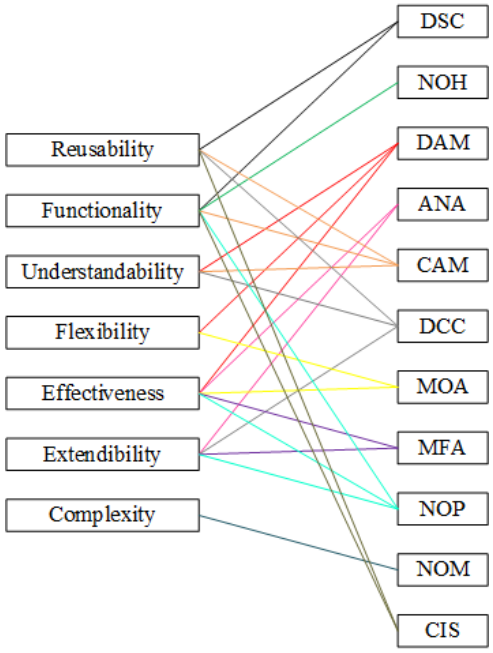


Fig. 3. Relationship of C&K metrics with software quality attributes.

In below, their background information and description has been given:

- DSC (Design Size in Classes): DSC is used to evaluate the design properties of a software. It has a good impact on reusability and functionality of a system.
- NOH (Number of Hierarchies): Like DSC, NOH is used to measure the design properties of a system. It is good to track the functionality of the software.
- DAM (Data Access Metric): DAM is used to control the variables, fields, or data names in the classes. It includes encapsulation design property so it is effective for understandability, flexibility, and effectiveness of a software.
- ANA (Average Number of Ancestors): ANA is used to measure the not only generalization aspect of a design but also the specialization. Effectiveness and extendibility of a software can be tracked with this metric.
- CAM (Cohesion Among Methods of Class): CAM is used to quantify the cohesion of a design property. It has a good impact on reusability, functionality, and understandability.
- DCC (Direct Class Coupling): DCC helps the coupling property of a design since it is used to measure classes which are directly relevant to other classes. Lower DCC value means the software understandability, reusability, and extendibility is high.

- MOA (Measure of Aggregation): MOA is used to quantify the aggregation of a design property. It effects the composition of class so it is good for flexibility and effectiveness of a system.
- MFA (Measure of Functional Abstraction): MFA is used to evaluate the inheritance of a system. Inheritance is good for extendibility, and effectiveness thus MFA is used to determine whether the software is effective and extendible or not.
- NOP (Number of Polymorphic Methods): NOP is used to asses polymorphism design property of a system. Polymorphism effects the extendibility, functionality, effectiveness, and flexibility of a software.
- NOM (Number of Methods): NOM is used to measure the complexity of a system. It is developed based on WMC metric from CK metrics suite.
- CIS (Class Interface Size): CIS is used to measure the messaging property of a software. Messaging is good for re-usability, and functionality.

Table 1 shows the metrics for each metrics suite.

TABLE I
POPULAR METRIC SUITES AND METRICS USED IN FAULT PREDICTION STUDIES.

| Ref. | Metrics Suite | Design Metrics |
|---|---|---|
| [6], [7] | CK | WMC, DIT, NOC, CBO, RFC, and LCOM |
| [8], [19], [20] | MOOD | MHF, AHF, MIF, AIF, CF, and PF |
| [9] | QMOOD | DSC, NOH, DAM, ANA, CAM, DCC, MOA, MFA, NOP, NOM, and CIS |

## III. ACQUISITION OF FAULT METRICS

In the previous section metric suites and their metrics have been mentioned. In this section, we provided the calculation of these metrics for each suite.

### *A. Acquisition of Chidamber and Kemerer metrics*

Calculation for each metrics has given in below:

- WMC: It is obtained by calculating the complexity of defined methods (local) in a class. If complexities are the same, it can be obtained by counting the number of methods of each class.
- DIT: It is obtained by counting the number of edges between the given class and a root class in an inheritance tree. If it is a root or does not have any base class the value of DIT will be 0.
- NOC: It is obtained by counting the number of direct children of a given class in an inheritance tree. If a given class has no children means the NOC value is 0.
- CBO: It is obtained by counting the number of classes which is used in a given class plus by counting the number of classes which used the variables or methods of a given class.
- RFC: It is obtained by counting the number of all local methods of a given class plus by counting the number of methods of other classes which are directly called from any of the methods on a given class.
- LCOM: It is obtained by counting the number of methods which have zero similarity minus by counting the number of methods which have non-zero similarity.

### *B. Acquisition of Abreu and Carapuca metrics*

Calculation for each metrics has given in below:

- MHF: It is obtained by counting the number of invisible methods from all classes by dividing by counting the number of methods in the system. Invisibility means the percentage of all classes which have invisible methods. If a method is private then invisible value will be 1 yet public makes this value 0. On the other hand, protected methods are calculated by the size of inheritance tree by dividing all classes.
- AHF: It is obtained by counting the number of invisible attributes from all classes by dividing by counting the number of attributes in the system. It has the same properties with MHF with respect to invisible calculation.
- MIF: It is obtained by counting the number of inherited methods by dividing the number of all methods defined in a given class.
- AIF: It is obtained by counting the number of inherited attributes by dividing the number of all attributes defined in a given class.
- CF: It is obtained by counting the number of non-inheritance couplings dividing by the maximum number of couplings in the software. Coupling factor generally ranges between 0 and 1.
- PF: It is obtained by counting the number of overrides by dividing the maximum number of possible method overrides.

### *C. Acquisition of Bansiya and Davis metrics*

Calculation for each metrics has given in below:

- DSC: It is obtained by counting the total number of classes in a given design.
- NOH: It is obtained by counting the number of hierarchies in a given design.
- DAM: It is obtained by rationing the number of private and protected attributes to the total number of attributes in a given class. It is essential to get DAM value high [9].
- ANA: It is obtained by rationing the number of classes inherited from a root class to the all classes.
- CAM: It is obtained by counting the summation of the intersections of parameters given in a class with the maximum independent parameters. The ideal CAM value should be close to the 1 [9].
- DCC: It is obtained by counting the number of different classes which is directly associated to the other classes.
- MOA: It is obtained by counting the total number data declarations which are defined by the user.

- MFA: It is obtained by rationing the number of inherited methods in a given class to the number of methods defined in a given class.
- NOP: It is obtained by counting the polymorphic methods in a system.
- NOM: It is obtained by counting the number defined methods in a given class.
- CIS: It is obtained by counting the number of public methods in a given class.

## IV. Conclusion

In this study, we analyzed the software metrics suites which are used software fault prediction studies. It is essential to predict the software faults before releasing process or in early development cycle since testing and fixing the software is highly expensive with regards to time and cost. It consumes nearly all the resources of a project. There are many metrics available today yet only the three most popular metrics (CK, MOOD, and QMOOD) were examined with respect to their design metrics and calculating steps. In section 2, each metric suite was studied and their definition was also provided. In Section 3, we provided the calculation process for each design metric. Software fault prediction and fault proneness is a popular topic in the software area and there are many studies exist to predict faults earlier based on machine learning algorithms. For these studies, researchers need to collect their own data if they do not want to use existing datasets thus this work is intended to guide the researchers on how to collect these data in their studies. Suggestions for researches in order to provide more successful software projects and to lower the effect of software faults and risks has given below:

- Before starting the software projects, researches should determine the best development approach for the given system based on the requirements. In small projects waterfall model is quite beneficial while prototype model is quite efficient on complex and large systems [25], [26]. There are many models exist for developing a software thus, researches should perceive the exact model.
- Incorporate with the clients during software development has good effects on software systems. This makes the development process more robust, reliable, secure and the system will be less tendency against unexpected requirements.
- In some cases, unreliable requirements, changing environmental factors, cultural differences, and lack of communication make the system more complex, and induce estimation of process and cost management and planning difficult [27]. In order to overcome these problems, project managers should track and monitor the causes of lessening distress.
- Risk management is a vital part of a project thus, project managers should develop a good risk management cycle. All risks should be categorized and should be estimated before they are happening, all resources should be used clearly and all risks should be resolved as early as possible.
- One of the most important things for software products is quality. It depends on many things and effects the development process in earlier stages. To make the system more quality all requirements should be collected clearly, the design should be planned in detail, test scenarios should be performed correctly, and documentation should be applied as early as possible [11].
- The software provides bugs, defects, faults and errors in any system. It is efficient to determine the testing approaches for each type of software faults. For software errors, error handling is far enough. It includes the process of detection and resolution of software errors (logical, generated, compile-time, and runtime). Fault management should be performed to specify the faults of the system including active and passive faults. With the development of machine learning algorithms, nowadays software faults can be determined in the earlier processes.

It is possible to develop an error-free software system in theoric. Yet all system perform errors or faults even researchers or project managers applied all these approaches. The important thing is developers should comprehend the systems do not stop giving errors and should perform the appropriate risk, test, and process management approaches.

## References


[1] G. J. Pai and J. B. Dugan, "Empirical Analysis of Software Fault Content and Fault Proneness Using Bayesian Methods," 2007.

[2] R. Malhotra and A. Jain, "Fault Prediction Using Statistical and Machine Learning Methods for Improving Software Quality," *Journal of Information Processing Systems*, vol. 8, no. 2, pp. 241–262, 2012.

[3] M. E. R. Bezerra, A. L. I. Oliveira, and S. R. L. Meira, "A Constructive RBF Neural Network for Estimating the Probability of Defects in Software Modules," in *Proceedings of International Joint Conference on Neural Networks*, pp. 2869–2874, 2007.

[4] Boehm B. W., "Industrial Software Metrics: A Top-Ten List," *IEEE Software*, vol. 4, no. 5, pp. 84–85, 1987.

[5] S. Demirel and R. Das, "Software requirement analysis: Research challenges and technical approaches," in *6th International Symposium on Digital Forensic and Security, ISDFS 2018 - Proceeding*, vol. 2018-Janua, 2018.

[6] S. R. Chidamber and C. F. Kemerer, "Towards a Metrics Suite for Object Oriented Design," in *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications*, pp. 197–211, 1991.

[7] S. R. Chidamber and C. F. Kemerer, "A Metrics Suite for Object Oriented Design," *IEEE Transactions on Software Engineering*, vol. 20, no. 6, pp. 476–493, 1994.

[8] F. Brito e Abreu and R. Carapuça, "Object-Oriented Software Engineering: Measuring and Controlling the Development Process," in *Proceedings of 4th International Conference on Software Quality*, no. October, pp. 3–5, 1994.

[9] J. Bansiya and C. G. Davis, "A Hierarchical Model for Object-Oriented Design Quality Assessment," *IEEE Transactions on Software Engineering*, vol. 28, no. 1, pp. 4–17, 2002.

[10] G. Gurgoze, R. Das, and I. Turkoglu, "Comparison of software development process models," in *The 8th International Advanced Technologies Symposium (IATS-2017)*, pp. 1923–1932, Frat Üniversitesi, 2017.

[11] T. B. Alakus, R. Das, and I. Turkoglu, "Yazilim gelitirme süreçlerinin analizi: Zorluklar, tasarim prensipleri ve tekniksel yaklaimlar," in *2017 International Artificial Intelligence and Data Processing Symposium (IDAP)*, pp. 1–10, IEEE, sep 2017.

[12] I. Vessey and R. Weber, "Research on Structured Programming: An Empiricist's Evaluation," *IEEE Transactions on Software Engineering*, vol. SE-10, no. 4, pp. 397–407, 1984.

[13] E. J. Weyuker, "Evaluating Software Complexity Metrics," *IEEE Transactions on Software Engineering*, vol. 14, no. 9, pp. 1357–1365, 1988.

[14] Y. Wand and R. Weber, "On the Deep Structure of Information Systems," *Information Systems Journal*, vol. 5, no. 3, pp. 203–223, 1995.
[15] C. F. Kemerer, "Reliability of Function Points Measurement," *Communications of the ACM*, vol. 36, no. 2, pp. 85–97, 1993.
[16] G. Singh, D. Singh, and V. Singh, "A Study of Software Metrics," *International Journal of Computational Engineering & Management*, vol. 11, pp. 22–27, 2011.
[17] A. K. Sharma, A. Kalia, and H. Singh, "Metrics Identification for Measuring Object Oriented Software Quality," *International Journal of Soft Computing and Engineering*, vol. 2, no. 5, pp. 255–258, 2012.
[18] U. Erdemir, U. Tekin, and F. Buzluca, "Nesneye Dayal Yazlm Metrikleri ve Yazlm Kalitesi," in *Yazlm Kalitesi ve Yazlm Gelitirme Araçlar Sempozyumu*, 2008.
[19] F. Brito e Abreu and W. Melo, "Evaluating the Impact of Object-Oriented Design on Software Quality," in *Proceedings of the 3rd International Software Metrics Symposium*, pp. 90–99, 2002.
[20] F. Brito e Abreu, "Design Metrics for Object-Oriented Software Systems," tech. rep., 2005.
[21] S. C. Misra and V. C. Bhavsar, "Measures of Software System Difficulty," *Software Quality Professional*, vol. 5, no. 4, pp. 1–16, 2003.
[22] J. J. Al-Ja'afer and K. E. M. Sabri, "Metrics For Object Oriented Design (MOOD) To Assess Java Programs," *King Abdullah II School for Information Technology*, pp. 1–9, 2004.
[23] K. K. Aggarwal, Y. Singh, A. Kaur, and R. Malhotra, "Empirical Study of Object-Oriented Metrics," *Journal of Object Technology*, vol. 5, no. 8, pp. 149–173, 2006.
[24] N. Jayalakshmi and N. Satheesh, "Software Quality Assessment in Object Based Architecture," *International Journal of Computer Science and Mobile Computing*, vol. 3, no. 3, pp. 941–946, 2014.
[25] R. Marques, G. Costa, M. Silva, and P. Gonalves, "A survey of failures in the software development process," in *Proceedings on 25th European Conference on Information Systems (ECIS)*, pp. 2445–2459, 2017.
[26] V. Rastogi, "Software Development Life Cycle Models - Comparison, Consequences," *International Journal of Computer Science and Information Technologies*, vol. 6, no. 1, pp. 168–172, 2015.
[27] W. Kobitsch, D. Rombach, and L. Feldman, Raimund, "Outsourcing in India," pp. 78–86, 2001.