International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 4 Issue 08, August-2015

Software Quality Measurement Through Testing

Dr. S. A. Sahaya Arul Mary
Dean Academics,
Jayaram College of Engineering & Technology,
Trichy.

M. Kavitha
Research Scholar, Dept. of Computer Science
Manonmaniam Sundaranar University,
Tirunelveli.

Abstract: **Software complexity is measured by various metrics of software. Software metrics play an important role in analyzing and improving software quality. Some of the metrics like reliability, reusability, size, etc are proposed by early researchers and they were very useful in software quality measurement. Software measurement faces a number of challenges whose solution requires both innovative techniques and borrowings from other disciplines. Over the years, a number of techniques and measures have been proposed and accessed via theoretical and empirical analyses. This shows the theoretical and practical interest of the software measurement field, which is constantly evolving to provide new, better techniques to support existing and more recent software engineering development methods. Software metrics are often categorized into products and process metrics. The aim of this paper is to discuss and analyse the various metrics which is used for software quality measurement proposed by early authors in this field. And also find a new metrics to improvise the Network Oriented Software Quality Measurement process.**

Keywords-- Software Metrics; Quality Measurement; Metric; Network Applications; OOS;

I. INTRODUCTION

Software is the sole of Microprocessor enabled devices. The effective utilization of hardware components are always decided by the quality of software. When talking about the quality, it is must to follow it in every stage of software. To find the quality of software, some of the metrics are defined by researchers early in the field of quality management. The first software metrics were proposed in the mid 70s. After the first proposal a large number of metrics have been proposed in the following years. The more number of metrics was followed by more practical proposals to find the results interpretation techniques from metrics [1]; all those metrics are divided into two major types as listed below,

- Internal product metrics: Measure attributes of the product that can be measured directly by examining the product on its own irrespectively on its behavior.
- External product metrics: Measure attributes of the product that can be measured only with respect to how the product relates to its environment.

Gurudev Singh et. al. [2] discussed the two types of Metrics, *Process Metrics*: Process metrics are known as management metrics and used to measure the properties of the process which is used to obtain the software. Process metrics include the cost metrics, efforts metrics, advancement metrics, and reuse metrics. Process metrics help in predicting the size of final system & determining whether a project on running according to the schedule.

Products Metrics: Product metrics are also known as quality metrics and is used to measure the properties of the software. Product metrics includes product non reliability metrics, functionality metrics, performance metrics, usability metrics, cost metrics, size metrics, complexity metrics and style metrics. Products metrics help in improving the quality of different system component & comparisons between existing systems. The various metrics are discussed detailed in forthcoming sections.

The aim of proposing metrics are to find the Software Quality Measurement. Various authors defined Measurement as below.

Formally, measurement is defined as a mapping from the empirical world to the formal, relational world. Consequently, a measure is the number or symbol assigned to an entity by this mapping in order to characterize an attribute [3].

"Measurement is the assignment of numbers to objects or events according to rule. [4] The rule of assignment can be any consistent rule. The only rule not allowed would be random assignment, for randomness amounts in effect to a non rule." [5].

"Measurement is the process of empirical, objective, assignment of numbers to properties of objects or events of the real world in such a way as to describe them." [6].

"Measurement is the process by which numbers or symbols are assigned to attributes of entities in the real world in such a way as to characterize them according to clearly defined rules." [7].

Measurement is "the act or process of assigning a number or category to an entity to describe an attribute of that entity." [8].

The Advantages of Software Metrics are extends to find the critical study of various programming language and characteristics of them, to perform comparative study of design methodologies, to compare the capabilities of productivity of people, to calculate the effort to be put in the design of the software, to check the complexity of code, etc.

The distribution of object-oriented paradigm has led to the need of cognitive complexity metrics. Several object-oriented metrics have been proposed in recent years. In order for those metrics to be usable metrics validation are needed [9].

Even having advantages there are also some limitation in software matrices. This literature Study is done to conclude the Roll of different software testing metrics in the area of product's quality measurement and its limitations.

Software Technology is a growing field and its growing level is too high than any other field. The metrics proposed before the invention of new technology never satisfies the

requirement of new software products produced by new technologies. Michalis Xenos et. al. had focused on product metrics and on how such metrics can aid in design, prediction and assessment of the final product quality, provide data used for decision-making, cost and effort estimation, fault prevention, testing time reduction and, consequently, aid in producing better software for E-Government and E-Commerce systems and it is summarized that metrics are an important instrument for the development of software to be integrated into E-Government and E-Commerce systems; metrics aid in making estimations in the early phases of a project, preventing problems in intermediate phases and evaluating quality in the late project phases. [10].

Now a day's applications are all converted into standalone mode to web mode. So the network applications are growing. The Metrics to qualify the good web applications should be derived.

Further in Section 2 some of the Metrics are discussed, in Section 3 the challenges in software Measurement are discussed and in section 4 the roll of metrics are concluded.

II. LIST OF METRICS

When discussing about software metrics its nature that the visible metrics come into the focus first. The metrics listed below are most common for software measurement.

- File Level Metrics
- Class Level Metrics
- Method/Function Level Metrics

All the above Metrics are calculated using the Lines of codes written, Number of variables declared, number of methods, constructor, functions used, number of statements used and type of statements etc.

Other Object Oriented Metrics are calculated based on the above basic metrics. Chidamber et. al., proposed some of the objected oriented metrics that are very famous in and important. The metrics proposed are Weight 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 of Methods (LCOM). These Metrics are called as CK cognitive Metrics [11].

Weighted Class Complexity (WCC), Extended Weighted Class Complexity (EWCC), Attribute Weighted Class Complexity (AWCC), Cognitive Weighted Response for Class (CWRFC) and Cognitive Weighted Coupling Between Object (CWCBO) are known as cognitive Metrics. All the above metrics are known as Cognitive Complexity Metrics (CCMS).

Both CK cognitive and cognitive metrics are objected oriented and they are calculated based on the basic metrics.

A. Weight Method per Class (WMC):

The understandability and reusability of software is decided by this metric.

A class is a template from which objects can be created. Classes with large number of methods are likely to more application specific, limiting the possibility of reuse. This set of objects shares a common structure and a common behavior manifested by the set of methods.

The WMC is a count of the methods implemented within a class or the sum of the complexities of the methods. But the second measurement is more difficult to implement because not all methods are accessible within the class hierarchy because of inheritance.

The larger the number of methods in a class is the greater the impact may be on children, since children inherit all of the methods defined in a class.

B. Response for Class (RFC):

A message is a request that an object makes to another object to perform an operation. The operation executed as a result of receiving a message is called a method.

The RFC is the total number of all methods within a set that can be invoked in response to message sent to an object. This includes all methods accessible within the class hierarchy.

This metrics is used to check the class complexity. If the number of method is larger that can be invoked from class through message than the complexity of the class is increase.

C. Lack of Cohesion of Methods (LCOM)

Cohesion is the degree to which methods within a class are related to one another and work together to provide well bounded behavior.

LCOM uses variable or attributes to measure the degree of similarity between methods. We can measure the cohesion for each data field in a class; calculate the percentage of methods that use the data field. Average the percentage, then subtract from 100 percent. Lower percentage indicates greater data and method cohesion within the class. High cohesion indicates good class subdivision while a lack of cohesion increases the complexity.

D. Coupling between Object Classes (CBO)

Coupling is a measure of strength of association established by a connection from one entity to another. Classes are couple in three ways. One is, when a message is passed between objects, the object are said to be coupled. Second one is, the classes are coupled when methods declared in one class use methods or attributes of the other classes.

Third one is, inheritance introduced significant tight coupling between super class and subclass. CBO is a count of the number of other classes to which a class is coupled. It is measured by counting the number of distinct non inheritance related class hierarchy on which a class depends. Excessive coupling is detrimental to modular design and prevent reuse. If the number of couple is larger in software than the sensitivity will affects the other parts of design.

E. Depth of Inheritance Tree (DIT)

Inheritance is a type of relationship among classes that enables programmers to reuse previously defined objects, including variables & operators. Inheritance decrease the complexity by reducing the number of operations and operators, but this abstraction of objects can make maintenance and design more difficult.

Depth of class within the inheritance hierarchy is the maximum length from the class node to the root of the tree, measured by the number of ancestor classes.

The deeper a class within the hierarchy, the greater the number of methods and is likely to inherit, making it more complex to predict its behavior. A support metric for DIT is the number of methods inherited.

F. Number of Children (NOC)

The number of children is the number of immediate subclasses subordinates to class in the hierarchy. The greater the number of children, the greater the parent abstraction. The greater the number of children, greater the reusability, since the inheritance is a form of reuse. If the number of children in class is larger than it require more testing time for testing the methods of that class.

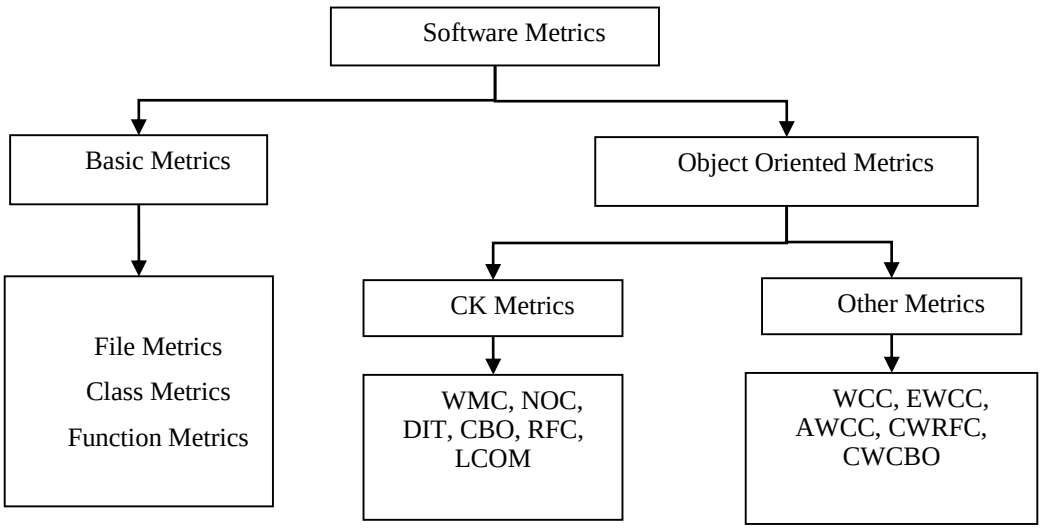

Fig. 1 List of Various Metrics

TABLE 1
SHOWS THE COMPARISON OF USAGE OF METRICS IN VARIOUS SOFTWARE COMPONENTS

Metrics	*Basic Metrics*	Object Oriented				
		Class	*Inheritance*	*Couplings*	*Response*	*Method Cohesion*
File Level Metrics	Yes	No	No	No	No	No
Class Level Metrics	Yes	Yes	No	No	No	No
Method/Function	Yes	No	No	No	No	No
WMC	Yes	Yes	No	No	No	No
DIT	Yes	Yes	Yes	No	No	No
NOC	Yes	Yes	Yes	No	No	No
CBO	Yes	Yes	Yes	Yes	No	No
RFC	Yes	Yes	Yes	Yes	Yes	No
LCOM	Yes	Yes	Yes	Yes	Yes	Yes
WCC	Yes	Yes	Yes	No	No	No
EWCC	Yes	Yes	Yes	Yes	No	No
AWCC	Yes	Yes	Yes	Yes	Yes	Yes
CWRFC	Yes	Yes	Yes	Yes	Yes	Yes
CWCBO	Yes	Yes	Yes	Yes	Yes	Yes

G. Weighted Class Complexity (WCC)

This metrics is proposed by Mishra[12] by modifying CC metric. This metrics is calculated by assuming that the class is a set of data and set of method accessing them. So the complexity of the class is measured by the complexity of methods and the attributes. The formula given below is designed by them to calculate WCC.

$$\mathrm{WCC} = Na + \sum_{p=1}^{n} \mathrm{MCp} \qquad (1)$$

Where,

N_a is the Number of Attribute
MC is the Method Complexity

H. Extended Weighted Class Complexity (EWCC)

Arockiam et. Al. [13] proposed this metrics by extending the WCC metrics. EWCC is the sum of weights of attributes and methods of the class and derived class. This metrics includes the cognitive complexity due to Inheritance. The below formula is used to find EWCC.

$$\mathrm{EWCC} = Na + \sum_{i=1}^{n} \mathrm{MCi} + \sum_{j=1}^{m} \mathrm{ICCj} \qquad (2)$$

Where,

Na is the total Number of attributes,
MC is the method complexity,
ICC is the inherited class complexity which is calculated by the below given formula

$$\mathrm{ICC} = (DIT\ x\ Cl)\ x \sum_{i=1}^{n} \mathrm{RMCi} + \mathrm{RNa} \qquad (3)$$

Where,

N is the number of inherited methods
RNa is the reused method complexity
ICC is the inherited class complexity
DIT is the Depth of Inheritance Tree
CL is the cognitive complexity of Lth level

I. Attribute Weighted Class Complexity (AWCC)

This is derived to calculate the complexity of class using Method complexity, attribute complexity and the inherited complexity. This is also the extension of WCC. This metrics is proposed by L. Arockiam et. al. [14]. The below formula is used to calculate AWCC.

$$AWCC = \sum_{i=1}^{n} ACi + \sum_{j=1}^{m} MCj + \sum_{k=1}^{m1} ICCk \quad (4)$$

Where,

AC is the attribute complexity
MC is the method complexity
ICC is the inherited class complexity

Attribute complexity is calculated by using the following formula

$$AC = (PDT * W_b) + (DDT * W_d) + (UDDT * W_u) \quad (5)$$

Where, PDT is the number of Primary Data Type attributes,
DDT is the Number of Derived Data Type attributes,
UDDT is the Number of User Defined Data Type attributes,
W_b is the cognitive weights of the PDT attributes,
W_d is the cognitive weights of the DDT attributes,
W_u is the cognitive weights of the UDDT attributes,

J. Cognitive Weighted Response for a Class (CWRFC)

A metric namely Cognitive Weighted Response For a Class (CWRFC) proposed by Aloysius et al.[14] In CWRFC, the cognitive weights are assigned to the function call statement based on the effort needed to understand their type of function calls due to message passed by an object of that class. CWRFC is used to calculate the complexity of the class using the Response Set complexity. If there are m numbers of response sets in a class, then the CWRFC of that class can be calculated by using the below formula.

$$CWRFC = \sum_{j=1}^{m} RSCj \quad (6)$$

Where, RSC is the response set complexity, which can be calculated as below,

$$RSC = M + \forall i Ri \quad (7)$$

Where M is set of all methods, R is set of methods called by any of those methods and it is calculated as below

$$R = DF * (CW_t + WF_d) + PBV * (CW_t + WF_v) + PBR * (CW_t + WF_r) \quad (8)$$

Where, DF is the total number of default function call
PBV is the total number of pass by value function call
PBR is the total number of pass by reference function call
CW_t is the Cognitive weights of the function call
WF_d is the Weighting Factor of the DFC statements,
WF_v is the Weighting Factor of the PBV statements,
WF_t is the Weighting Factor of the PBR statements

K. Cognitive Weighted Coupling Between Object (CWCBO)

Aloysius et al. [14] proposed a metric called cognitive weighted coupling between objects (CWCBO). Which considers the cognitive complexity of the different types of coupling such as data coupling, control coupling, global coupling and interface coupling, "unnecessary object coupling needlessly decreases the reusability of the coupled objects", " Unnecessary object coupling also increases the chances of system corruption when changes are made to one or more of the coupled objects. The following formula is designed by them.

$$CWCBO = ((CC*WFCC) + (GDC*WFGDC) + (IDC*WFIDC) + DC*WFDC) + (LCC*WFLCC) \quad (9)$$

CC is the total number of modules that contains control coupling,
GDC is the count of Global Data coupling
IDC is the count of internal data coupling
DC is the count of Data Coupling
LCC is Count of lexical Content Coupling

WFCC is the Weighting factor of control coupling
WFGDC is the Weighting factor of Global Data Coupling and its weight is given as 1
WFIDC is the Weighting factor of internal Data coupling and its weight is given as 2.
WFDC is the weighting factor of data coupling and its weight is given as 3
WFLCC is the weighting factor of lexical content coupling and its weight is given as 4.

III. CHALLENGES IN SOFTWARE MEASUREMENT

Software measurement poses a number of challenges, from both a theoretical and practical points of view. To face these challenges, we can use a number of techniques that have been developed over the years and/or have been borrowed from other fields.

First, we need to identify, characterize, and measure the characteristics of software processes and products that are believed to be relevant and should be studied. This is very different from other engineering branches, where researchers and practitioners directly use measures without further thought. In those disciplines, there no longer is a debate on what the relevant characteristics are, what their properties are, and how to measure these characteristics. In software engineering measurement, instead, we still need to reach that stage. There is not as much intuition about software product and process characteristics (e.g., software cohesion or complexity) as there is about the important characteristics of other disciplines. Therefore, it is important that we make sure that we are measuring the right thing, i.e., it is important to define measures that truly quantify the characteristic they purport to measure. This step—called theoretical validation—is a difficult one, in that it involves formalizing intuitive ideas around which there is limited consensus. To this end, one can use Measurement Theory,

which has been developed in the social sciences mainly in the last 60 years, or property-based approaches, which have been used in Mathematics for a long time.

Second, we need to show that measuring these characteristics is really useful, via the so-called empirical validation of measures. For instance, we need to show if and to what extent these characteristics influence other characteristics of industrial interest, such as product reliability or process cost. It is worthwhile to measure them and use them to guide the software development process only if they have a sufficiently large impact. To this end, experiments must be carried out and threats to their internal and external validity must be carefully studied.

In addition, the identification and assessment of measures may not be valid in general. Nothing guarantees that measures that are valid and useful in one context and for some specified goal are as valid and useful for another context and goal. Goal oriented frameworks that have been defined for software measurement can be used.

IV. CONCLUSION

Delivering a quality product is the key axiom of every software development team. To make the product quality it is must the measure the quality of software by using strong valid well furnished metrics. In this paper some of the early proposed metrics are discussed. But no metrics is useful to check the quality measurement of network oriented applications. From the survey made on this paper it is concluded that researcher should kick start to derive the network oriented metrics in future.

REFERENCES

[1] Shepperd, M., & Ince, D. (1990). The Use of Metrics in the Early Detection of Design Errors. Proceedings of Software Engineering.
[2] A Study of Software Metrics Gurdev Singh, Dilbag Singh, Vikram Singh IJCEM International Journal of Computational Engineering & Management, Vol. 11, January 2011 ISSN (Online): 2230-7893
[3] Cem Kaner,Senior Member,IEEE, and Walter P.Bond, "Software Engineering Metrics:Whay Do They Measure and How Do We Know?"
[4] S. S. Stevens, "On the Theory of Scales of Measurement," Science, vol.103, pp. 677-680, 1946.
[5] S.S.Stevens,Psychophysics: Introduction to its Perceptual, Neural, and Social Prospects. New York: John Wiley & Sons, 1975.
[6] L. Finkelstein, "Theory and Philosophy of Measurement," in Theoretical Fundamentals, vol. 1, Handbook of Measurement Science, P. H. Sydenham, Ed. Chichester: John Wiley & Sons, 1982, pp.1-30.
[7] N. E. Fenton and S. L. P fleeger, "Software Metrics: A Rigorous and Practical Approach," 2nd Edition Revised ed. Boston: PWS Publishing, 1997.
[8] IEEE, "IEEE Std. 1061-1998, Standard for a Software Quality Metrics Methodology, revision." Piscataway, NJ,: IEEE Standards Dept., 1998.
[9] Hericko, M, Rozman. I, Horvat. R, Domjinko. T, Gyorkos. J, "OO Metrics data gathering enrironment, in: Proceedings of Technology of Object Oriented Languages" (Tools 24), 1998, pp.80-85.
[10] Michalis Xenos,"Software Metrics and Measurements", In "Encyclopedia of E-Commerce, E-Government and Mobile Commerce", Mehdi Khosrow-Pour (Ed.), Idea Group Publishing, ISBN: 1-59140-799-0, pp. 1029-1036, 2006.
[11] Chidamber. S, Kemerer. C, "A Metrics Suite for Object Oriented Design", IEEE Transaction on Software Engineering 20(6) (1994) 476-493.
[12] Sanjay Misra and k. Ibrahim Akman, "Weighted Class Complexity: A Measure of Complexity for Object Oriented System," Journal of Information Science and Engineering, 2008, pp.1689-1708.
[13] L. Arockiam, A. Aloysius and J. Charles selvaraj, "Extended Weighted Class Complexity: A new of software complexity for objected oriented systems", Proceedings of International Conference on Semantic Ebusiness and Enterprise computing (SEEC), 2009, pp. 77-80.
[14] A. Aloysius, L. Arockiam, "Maintenance effort prediction model using cognitive complexity metrics", International Journal of Advanced Research in computer Science and Software Engineering, Vol 3, issue 11 November 2013.