ISSN:0975-9646

On the reliance of COM Metrics for a C# Project

Harsh Bhasin[#], Deepika Sharma[*], Rashmi Popli[^]

[#]*Assistant Professor, Department of Computer Science*
FMIT, Jamia Hamdard, New Delhi, India

[*]*M.Tech Scholar, Department of Information Technology*
YMCAUST, Faridabad, India

[^]*Assistant Professor, Department of Computer Engineering*
YMCAUST, Faridabad, India

***Abstract*— Can you judge the love of a person? His affection? His intentions? The answer is a big No!. However, the bank balance, the assets and even the number of research papers can be measured. Marty Rubin once said "Every line is the perfect length if you don't measure it." So, to every software developer his software would be just perfect if there is no metric to measure it. This premise compromises the reliability of software. Therefore it becomes immensely important to be able to measure software as well. In order to accomplish this task, software metrics come to our rescue. The present work analyzes various software metrics, puts them in right perspective and suggests the model to prioritize the metrics so that the software under development can be measured in a comprehensive way with lesser effort. The experiment conducted in order to prove the above premise has also been discussed in the paper. The paper also throws some light on the applicability of software metrics in C# projects.**

***Keywords*— Software Metrics, Object Oriented, COM, Reliability.**

I. Introduction

Software metrics are gaining importance day by day. They help us to enforce the same set of standards in software projects as physical sciences. The interest in software metrics dates back to the inception of the discipline of software engineering itself. Generally, they are perceived as an active measurement; however, some of the researchers have also considered software metrics as the major of degree to which a system, component or a process possess a given attribute [1]. Researchers have defined software metrics as per their importance to their respective field.

Software metrics have been used in various areas like in the determination of cost and size, in the prediction of quality levels and to provide quantitative check etcetera [2].

Software metrics can be compared to one of the branches of physics i.e. units and dimensions. As in the case of units, a unit must be precisely defined, it should be comparable and repeatable. For example, in order to define a second, we cannot take a year as a standard and say a second is $\frac{1}{31557600}^{th}$ of a year, as it should be defined in second but second cannot be defined in year. Moreover, this definition of second would not be precise. So, in order to overcome this problem, second has been defined in terms of the time taken by an electron to go from one level of Cesium (Cs) to another. 9192631770 such transitions constitute 1 second.

This definition is precise as measurements are done with the help of optical interferometer [3], it is repeatable because the laboratory of any city can measure 1 second using the instruments and is comparable also. In the same way we desire software metrics to be precise, comparable and repeatable. This is what is expected from software metrics as well.

This work examines the work of software metrics, puts them in right perspective and proposes a novel model of dealing with colossal number of metrics still, keeping the integrity of the software intact. The main goals of this paper are:

- To classify software metrics.
- To examine the importance of each metrics.
- To be able to present a model to prioritize metrics, especially for a C# project.

Rest of the paper has been organized as follows. Section two discusses the basics of software metrics, section 3 discusses the proposed model and the fourth section concludes and discusses the future scope.

II. Software Metrics

A. *Design Metrics*

The design generally includes use cases. In order to measure an Object Oriented Software, the number of actors and use cases are counted. As per the review the actors can further be segregated as simple, in case of an interface; complex, in case of an intricate interface or complex, in case of a graphical interface [4]. According to some researchers, the interactions in these diagrams are also subject to segregation. The interactions can also be simple, average or complex.

B. *Web Metrics*

A software developer must be able to measure the application he is developing. In case of web applications the following metrics come to our rescue.

- The number of static pages, in which the content does not change as such
- The number of dynamic pages, in which the content is updated from a database
- The number of links both internal links and the external links
- Some of the web sites, which are heavy on content, measure their size using number of words as well
- The other objects like videos, pictures, audio files etcetera also help to find the size of a page [5].

C. Process and Product Metrics

In software development process, what is being developed is product and how it is developed is a process. The efficiency of a product cannot be judged merely by considering the product but the process also needs to be considered in order to ascertain the efficiency of the system developed. Process and Product metrics come to our rescue in finding out the efficiency of the system by considering both the product and the process. As per the literature review, the process indicators take care of the status of the project. These indicators also help us to track risks and hence find out the problem area. Needless to say the whole thing helps in developing a quality product.

If these process metrics can be known to the team only, they are referred to as private process metrics. On the other hand, public process metrics help an organization to make strategic changes and to evaluate the performance of the concerned team. This can be done by finding out the number of errors and defects and identifying their corresponding cost.

The project metrics, on the other hand, are needed to avoid development schedule delays [6]. They are based on the input, output and results. These factors can, in turn, depend on the Lines of Code (LOC) and various other size metrics. It may also be noted that in such cases the Function Points (FP) and Object Oriented Metrics, discussed in the next sub-section, can also help us.

The goal of analyzing above two metrics should be, to map the metrics with the quality of the system being developed.

According to some authors, the product and the process metrics can also be internal or external. Those relating to the structure are external whereas those relating to the behavior are internal. Figure 1 summarizes the discussion.

D. Object Oriented Metrics

Object Oriented Metrics are used for measuring Object Oriented Software. This topic is widely researched and the list of these metrics has been presented in Table 1.

TABLE I Object Oriented Metrics

Sr. No.	Metric	Description	Ref. No.
1	Lines of Code (LOC)	This metric counts the lines of source code	7
2	Cyclomatic Complexity	It measures the number of independent paths through a program's source code.	8
3	Comment Percentage		9
4	Weighted Methods per Class (WMC)	WMC is a count of sum of complexities of all methods in a class.	10
5	Response for a Class (RFC)	It is number of methods in the set of all methods that can be invoked in response to a message sent to an object of a class.	10
6	Coupling Between Objects (CBO)	It is the measure of the average degree of connectivity and interdependency between objects in a model.	10
7	Lack of Cohesion in Methods (LCOM)	LCOM is the number of different methods within a class with reference to a given instance variable.	10
8	Number Of Children (NOC)	It is defined as the number of immediate subclasses.	10
9	Depth of Inheritance Tree (DIT)	It is defined as the maximum length from the node to the root of the tree and measured by the number of ancestral classes.	10
10	Method Hiding Factor (MHF)	This metric is the ratio of the total inherited methods and total methods defined.	11
11	Attribute Hiding Factor (AHF)	This metric is the ratio of hidden (private and protected) attributes to total attributes.	11
12	Method Inheritance Factor (MIH)	MIF is defined as the ratio of the sum of inherited methods in all classes of the system under consideration to the total number of available methods for all classes.	11
13	Attribute Inheritance Factor (AIF)	AIF is defined as the ratio of the sum of inherited attributes in all classes of the system under consideration to the total number of available attributes for all classes.	11
14	Coupling Factor (CF)	It is the ratio between the couplings and the maximum number of possible couplings among all the classes.	11
15	Polymorphism Factor (PF)	It measures the degree of method overriding in the class inheritance tree.	11
16	Data Abstraction Coupling (DAC)	DAC is the number of Abstract Data Types (ADTs) defined in a class.	12
17	Message Passing Coupling (MPC)	MPC is the number of send statements defined in a class.	12
18	Number of Methods Overridden by a subclass (NMO)	It counts the number of redefined methods in the class.	13
19	Reuse Ratio (RR)	The Reuse ratio is given by ratio between numbers of super classes by total number of classes.	13
20	Specialization ratio (SR)	The Specialization ratio is given as the ratio number of subclasses by number of super classes.	13
21	Maintainability Index (MI)	Maintainability Index is a software metric which measures how maintainable the source code is.	14

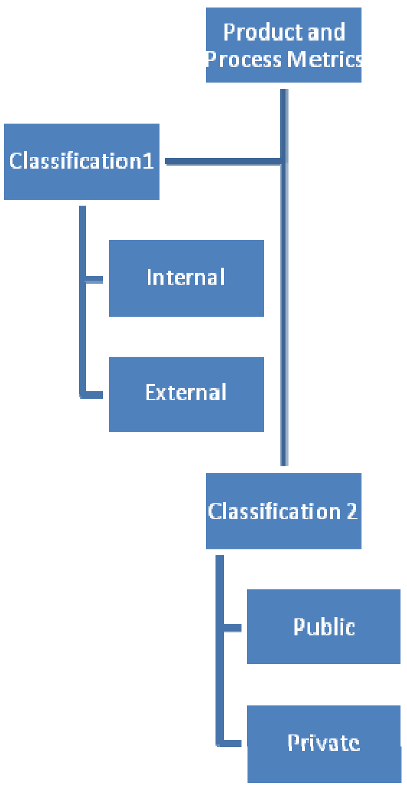

Fig. 1 Process and Product Metrics

E. *Metrics for a C# Project*

The metrics for a project developed in C#, in Visual Studio can be ascertained with the help of the many metrics. However, as per Microsoft [14], the following metrics are more important as compared to others.

1) Lines Of Code (LOC): The LOC indicates the number of lines in a module. It may be stated here that, the number of lines in case of a C# project is not same as the LOC of a C program. In the former case LOC is counted for a module whereas in the later case it is generally assessed for a whole program. The metric has traditionally been associated with the amount of work done and in the maintenance of the module. So, ideally if the value of LOC for a particular module is too high then it should be split into maintainable parts.

2) Depth of Inheritance (DIT): As per the literature review, this metric indicates extend to the root of the class hierarchy [10]. The problem with high depth is the possible intractability of the definition and the use of metrics.

3) Class Coupling: Coupling has been used as a credible metric by most of the researchers working in the discipline. However, as of now no credible study has analyzed the relevance of the types of coupling in case of Component Object Model (COM) projects developed in C#.

4) Cyclomatic Complexity: It generally refers to the number of independent paths that a program may have. The increase in the metric may indicate the increase in the test cases and plausible problem in dealing with all the paths.

5) Maintainability Index: It estimates an indicator value sandwiched between 0 and 100. This metric signifies the ease of maintaining the code [14]. If the value of this indicator is high it means better maintainability. Figure 2 shows the relationship between the value of the metric and maintainability.

🟥 **0-9:** Low Maintainability	🟨 **10-19:** Moderate Maintainability	🟩 **20-100:** Good Maintainability

Fig. 2 Relation between Maintainability Index and Maintainability

F. *Problems with Software Metrics*

Metrics are used for taking decision regarding effort, testing and so on. However, the decision would be incorrect if the basis itself is flawed. The metrics are crafted considering data to be normally distributed [15] which may not always be the case. Interestingly the data elements may lie outside the realm of the domain itself, referred to as outliers, also pose a big threat to the application of metrics.

It is also important to take care of the units of measurements and scale of the data, before considering them for problem which depends on various data samples. The error may lead to incorrect designs and hence irreparable damage.

III. Experiment

A. *Hypothesis*

For a professional C#, client side, application the same amount of quality can be achieved using lesser metrics.

B. *Methodology*

In order to verify the above hypothesis professional software developed in C# is taken. The software was management software of about 3K LOC. Three cycles of testing were carried out. The software was tested and the average APFD came out to be 62%. The testing was done using the 21 object oriented metrics, discussed earlier.

The number of metrics in the second phase was reduced to 5 in accordance with the discussion in section 2.4. The software again went to three cycles of testing. The test cases were crafted again in accordance with the new set of metrics. The APFD, in this case, came out to be 61.8, which is almost same as that of earlier phase.

The above experiment confirms the hypothesis stated earlier. The proposed technique is now being applied to bigger software of around 8K LOC. The point to be noted is that, as gains the common perception the measurement need not be more rather they should be affective. Moreover, many researchers stop at the maintainability part. This work on the other hand explores the use of measurements in testing as well. In one of our works the empirical relation between the two has been established. The results of the experiments being carried out are encouraging.

IV. CONCLUSIONS

There is a golden rule: “something is better than nothing”. The thing also holds for software metrics. As a software developer, even if you are not aware of the intricacies of the discipline of software metrics, you can at least start with the basic metrics like lines of code. This paper makes an attempt to introduce various metrics. Various software metrics have been discussed and analyzed in the preceding discussion. After understanding the concept it is important to decide which combination must be used to make the software as reliable as possible. However, it is important to put the things in the right perspective, especially in relation to the applications that the metrics might have in software development. The applications of the concept entails cost estimation, effort and many more things.

It may be noted that metrics give us a way forward to automation of testing, which is the ultimate desire of any developer. In one of our works effort has been made to develop an automated test generator via cellular automata [16]. In the extension of the work, test case generation was done via artificial life [17]. Effort is being made to use the concept of metrics in the automation process. Some of the researchers have emphasized on the fact that the productivity metrics, which tells us about developers and testers, must not be confused with the productivity metrics, which are concerned with the product.

The most important point is regarding the importance of metrics. It may be noted the metrics as such do not wear out. However, the importance attached to a metric may change [18]. For example the NOC metric discussed earlier will not hold the same importance in C#, as it does in C++. The software development technique in C# is radically different as compared to C++.

REFERENCES

[1] Y. Wang, Q. He, et. al.. 2002. Product and Process Metrics: A Software Engineering Measurement Expert System. 4th International Conference, PROFES 2002 Rovaniemi, Finland, December 9–11, 2002 Proceedings, 337 - 350.

[2] T. Gilb. *Software Metrics*, Winthrop Publishers, Inc., Cambridge, Massachusetts, 1977.

[3] Halliday & Resnick. Fundamentals of Physics, 3E, Wiley 1988.

[4] M. Sheppard. 1990 Design metrics: an empirical analysis. Sogtware Engineering Journal, vol. 5, no. 1.

[5] B. Lilburne, B. Devkota, K. Khan. 2004. Measuring Quality Metrics for Web Applications. Technical Report, IRMA International Conference, New Orleans, USA.

[6] R. Pressman. (2000). Software Engineering: a ractitioner's Approach: European Adaptation, 5th edition, McGraw-Hill, UK

[7] B. Londeix, *Cost Estimation for Software Development*, Addison Wesley, Reading, Massachusetts, 1987.

[8] T.J. McCabe, "A Complexity Measure," *IEEE Transactions on Software Engineering*, Vol. SE-2, No. 4, October 1976, pp. 243 - 245.

[9] D. Tegarden, S. Sheetz, and D. Monarchi. 1992. Effectiveness of Traditional Software Metrics for Object-Oriented Systems. Proceedings of the Twenty-Fifth Hawaii International Conference on System sciences, 359-368.

[10] S. Chidamber, and C. Kemerer. (1994). A Metrics Suite for Object Oriented Design, IEEE Transactions on Software Engineering, vol. 20, no. 6, pp. 476-493.

[11] R. Harrison, S. Counsell and R. Nithi, "An Evaluation of the MOOD Set of Object-Oriented Software Metrics," IEEE Trans. Software Eng., vol. 24, no. 6, pp. 491-496, June 1998.

[12] Li, W. and S. Henry, Object-oriented metrics that predict maintainability. Journal of Systems and Software, vol. 23(2): p. 111-122, 1993.

[13] M. Lorenz and J. Kidd, Object-Oriented Software Metrics, Prentice Hall, Englewood Cliffs, New Jersey, 1994.

[14] http://msdn.microsoft.com/en-us/library/bb385914.aspx

[15] J.M Bieman, N Fenton, D.A Gustafson, A Melton, L.M Ott, Fundamental issues in software measurement, A Melton (Ed.), Software Measurement, International Thompson Computer Press (1995), pp. 39 – 52

[16] H. Bhasin, and N. Singla. 2013. Cellular-Genetic Test Data Generation. ACM Sigsoft Software Engineering Notes, vol. 38, no. 5, 1-9.

[17] H. Bhasin., Shewani and D. Goyal. 2013. Article: Test Data Generation using Artificial Life. International Journal of Computer Applications, vol. 67 no.12, 34-39, Published by Foundation of Computer Science, New York, USA.

[18] M.S. Feather and T. Menzies. 2002. Converging on the optimal attainment of requirements. In IEEE Joint Conference On Requirements Engineering ICRE’02 and RE’02, 9–13th September, University of Essen, Germany.