AN APPROACH ORIENTED TOWARDS ENHANCING A METRIC PERFORMANCE

Rohitt Sharma
Department of Computer Science Engineering
Lovely Profesional University
Phagwara, India
rohittsharma@mailingaddress.org

Paramjit Singh
Department of Computer Science Engineering
Lovely Profesional University
Phagwara, India
paramjit.12969@lpu.co.in

Sumit Sharma
Department of Computer Science Engineering
Lovely Profesional University
Phagwara, India
sumit_sharma@mailingaddress.org

***Abstract*— Software Engineering like all other engineering professions has metrics and Software metrics are increasingly playing a central role in the planning and control of software development projects [2]. A software metric is a measure of some property of a piece of software or its specifications. These can be used to take meaningful and useful managerial and technical decisions related to cost, effort, time. Although the use of metric doesn't ensure the quality of a product but it can help in improving its quality. By analyzing the various research works, it is observed that the major work is either being done in comparing the metrics by applying it to various different projects or is in creating a new metric, which is the blend of former metrics, but nothing has been done to improve the metrics itself. This research will help the current organizations to improve the metric performance by analyzing the factors deduced in this research work.**

Keywords- Software Metrics, Static Analysis, Dynamic Analysis, Complexity.

I. INTRODUCTION

Metrics are used by the software industry to quantify the development, operation and maintenance of software. Software Metrics is a Measurement Based Technique which is applied to processes, products and services to supply engineering and management information and working on the information supplied to improve processes, products and services, if required[1]. They give us knowledge of the status of an attribute of the software and help us to evaluate it in an objective way. Thus, incorporating metrics into software development process is a valuable step towards creating better systems. The practice of applying software metrics to a software process and to a software product is a complex task that requires study of the discipline, and which brings knowledge of the status of the process and / or product of software in regards to the goals to achieve. Two of the time-honoured and popular software metrics have been Cyclomatic Complexity and LOC (lines of code)[7]. Originally, these measures were defined for the procedural programs and then in later stages they were incorporated for object-oriented systems. Cyclomatic complexity measures the amount of decision logic in a single software module[5] and LOC measures size of a module/program by counting the lines in the particular source code.

At the moment the major work in this field been done is either in comparing the metrics by applying it to various different projects or is in creating a new metric, which is the blend of former metrics, but, nothing much is been done in improving the metrics. We all know that a metric shows different results when applied in static

environment than when applied in dynamic environment, but, the factors behind it are not known yet. Thus, We are proposing an approach oriented towards analyzing the factors behind the cause of deviation of results.

The rest of the paper is organized as follows : Section 2 describes the Static and Dynamic analysis perspectives. Section 3 discusses about the research methodology. Section 4 details the experimentation and evaluation results with necessary tables and graphs and Section 5 concludes the paper along with the future work.

II. STATIC AND DYNAMIC ANALYSIS PERSPECTIVES

Analysis is about problems rather than solutions, discovery rather than invention, questions rather than answers, multiple perspectives rather than one. In Software Engineering analysis is basically done in following two ways: 1) Static Analysis 2) Dynamic Analysis.

A. Static Analysis

Static analysis, as by the name depicts, means the analysis of the code in static environment. Static code analysis, is a method of computer program debugging that is done by examining the code without executing the program[6]. It provides the basic understanding of the structure of the code and this can help in ensuring the code to be of industry standards. Static analysis involves no dynamic execution and can detect possible faults such as unreachable code, undeclared variables, parameter type mismatches, uncalled functions and procedures, possible array bound violations, etc.[8]. The compiler generated errors are basically the results of static analysis of that code, and information of these syntactical errors proves very valuable during maintenance and debugging work.

B. Dynamic Analysis

Dynamic analysis means the analysis of code in dynamic environment. A dynamic analysis measures the efficiency and effectiveness of software testing by monitoring the software system during its execution phase[4]. The objective is to find errors in a program while it is running, rather than by repeatedly examining the code offline. In order to implement it, a software system with operational version is necessary, and In static analysis, it should be taken care of that analysis only deals with the structure and development of the software source code, and no running code module must be there in it.

III. RESEARCH METHODOLOGY

After reading various research papers, the basic trend observed was of either proposing a hybrid metrics or creating a new metrics on a whole. Hybrid metrics focused at joining two or more highly efficient metrics, as in [9], various new metrics and their formulas are specified.

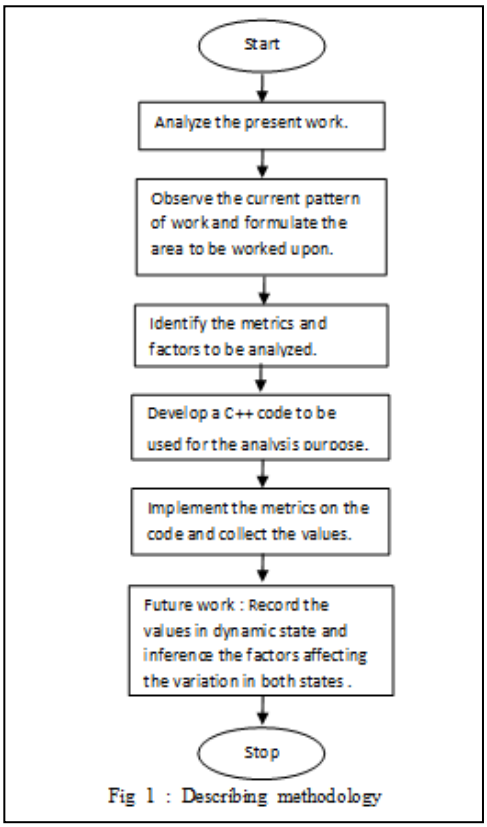


Fig 1 : Describing methodology

There are some other approaches also, that led towards creating a metric calculating tool to speed up the process of metrics value generation, as in [10], a new software metric analyzer is created, which eases the task of finding a particular metrics value. But in all the methodologies, there was no consideration on improving the metrics by any means, Hence, we are proposing a way of improving metrics performance.

The Fig 1 is self explanatory in terms of explaining the methodology to be followed in the way to propose a new approach towards enhancing the metric performance, and gives the brief idea of what is to be done.

IV. BIFURCATING FACTORS BASED ON EXPERIMENTAL RESULTS

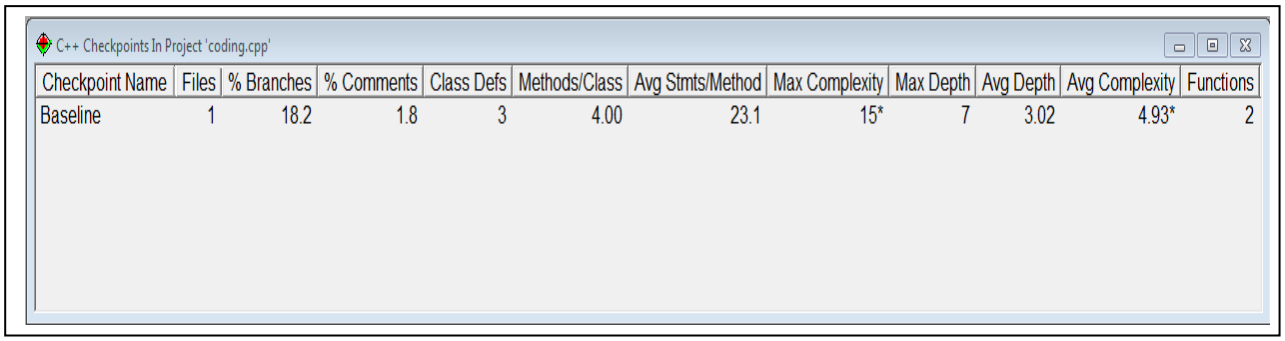
The developed c++ code is statically analyzed using two different tools and the values obtained after the process are as follows :

Metric	Tag	Overall	Per Module
Number of modules	NOM	5	
Lines of Code	LOC	480	96.000
McCabe's Cyclomatic Number	MVG	68	13.600
Lines of Comment	COM	10	2.000
LOC/COM	L_C	48.000	
MVG/COM	M_C	*6.800*	
Information Flow measure (inclusive)	IF4	4	0.800
Information Flow measure (visible)	IF4v	0	0.000
Information Flow measure (concrete)	IF4c	4	0.800
Lines of Code rejected by parser	REJ	0	

Fig 2: Results obtained from tool CCCC

From Fig 2, the values of first part of static analysis performed on code are recorded, which figures out the number of modules in the code to be 5, with 480 lines of code and McCabe's cyclometic complexity number to be 68 for the whole code. The values recorded in the second part of analysis are specified in figure 3.

Both the experiments conducted are on the same set of code generated the value of maximum complexity to be 15, maximum depth of statements to be 7, average depth to be 3.02 and average complexity is recorded. The results obtained was used to draw a kiviat graph to unriddle the interrelation between the various factors and a block histogram to understand number of statements with similar the block depth.


C++ Checkpoints In Project 'coding.cpp'

Checkpoint Name	Files	% Branches	% Comments	Class Defs	Methods/Class	Avg Stmts/Method	Max Complexity	Max Depth	Avg Depth	Avg Complexity	Functions
Baseline	1	18.2	1.8	3	4.00	23.1	15*	7	3.02	4.93*	2

Fig 3: Results obtained from tool SourceMonitor.

A Kiviat diagram is used to graphically represent and compare multiple entities and then to evaluate them against more than two variables, which gives it an edge over other charting and graphing techniques. and a Block Histogram allows us to see distribution of numeric values in a data set. The x-axis is divided into 'bins' that correspond to value ranges. Each item in the data set is drawn as a rectangular block, and the blocks are piled into the bins to show how many values in each range.

Table 1. Metrics details for the written code.

Parameter	Value
Project Name	CODING.cpp
Checkpoint Name	Baseline
Percent Lines with comments	1.8
Classes Defined	3
Methods Implemented per Class	4.00
Average Statements per Method	23.1
Line Number of Most Complex Method	118
Maximum Complexity	15
Line Number of Deepest Block	375
Maximum Block Depth	7
Average Block Depth	3.02
Average Complexity	4.93
Functions	2

Thus the kiviat graph in Fig 4 depicts the inter-relation between %comments, Average complexity, Average Depth, Maximum Depth, Maximum Complexity, Average Statements per Method and Methods per class, formed from the above values from Table 1 is as given below :

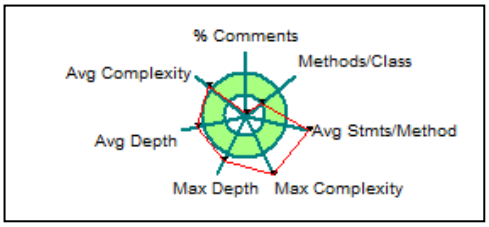


Fig 4: Kiviat graph

Table 2. Number of statements corresponding to the block depth

Block Depth	Statements
0	14
1	50
2	80
3	76
4	103
5	41
6	14
7	1
8	0

9+	0

The Table 2, provides the information of number of statements with a particular depth of block, like, number of statements having block depth of 4 are 103, etc. Hence formulated block-depth histogram in Figure 5 is as under :

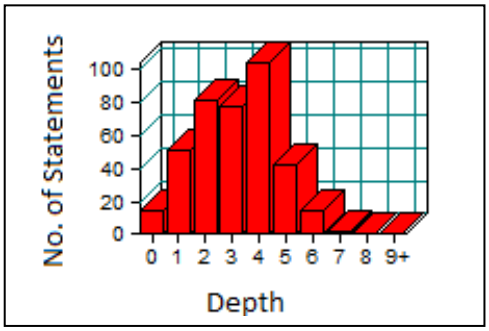


Fig 5: Block histogram of Statements vs Depth

After analyzing this code, a similar program with same line of code but with more number of modules and more number of comments was evaluated using the same tool and same process. But, there was a change in resulting values, the complexity factor of the second code was increased. Although the number of lines of code remained the same, but then even the results varied in both the cases, this can be due to increase in number of modules and lines of comments, that, there was an increase in complexity factors of the code. .

V. CONCLUSION

The above conducted experiment leads to the results that the changes made in code to enhance the functionality, do affects the other factors. Like in above experiment, number of modules was increased symbolizing the increase in to use the modular approach, the modular approach do increases the flexibility of the code and increasing the reusability, but the dynamic analysis of the code proves the add on the cost in terms of increased need of memory space and extra time for execution because some modules may partly repeat the task performed by other modules[3]. In same way, the line of comments do enhances the readability of the code and makes it more easy to understand the code for the developer after some period of time or for the third person. But it should be in some ratio as excessive use of LOC increases the compilation work as all the commented lines to be checked and skipped which do consumes a unit work of time. The other affected factor in above experiment was complexity. Now effects of increased complexity at the run time can be evaluated by profiling and figuring out the factors behind the deviation from static environment.

Hence, we propose the future work to generate the results in dynamic environment, and then perform the comparative analysis of the new results with the current obtained results of static state to find the factors that cause the deviation in performance in both the states and then to use those new factors to enhance the performance of the metrics.

REFERENCES

[1] Mr. Premal B. Nirpal & Dr. K. V. Kale (2011). A Brief Overview Of Software Testing Metrics, International Journal on Computer Science and Engineering (IJCSE), Vol. 3 No. 1 Jan 2011.
[2] S.Babu, R.M.S.Parvathi, (2011) “Implementation Of Dynamic Coupling Measurement Of Distributed Object Oriented Software Based On Trace Events” , IJCSEA, Vol.1, No.6.
[3] Vinod Dussad (2010) “Modular Programming”, http://www.indiastudychannel.com/resources/ 104906-Modular-Programming.aspx.
[4] Alice T. Lee, Todd Gunn, Tuan Pham, Ron Ricaldi, “Software Analysis Handbook: Software Complexity Analysis and Software Reliability Estimation and Prediction”, National Aeronautics and Space Administration, 1994.
[5] “Cyclomatic Complexity”,http://hissa.nist.gov/HHRFdata/Artifacts/ITLdoc/235/chapter2.htm.
[6] “static analysis (static code analysis)”, http://searchwindevelopment.techtarget.com/definition/ static-analysis.
[7] Chhabra JK, Gupta V, “A survey of dynamic software metrics”, Journal Of Computer Science And Technology, 25(5): 1016–1029 Sept. 2010. DOI 10.1007/s11390-010-1080-9.
[8] Webb, J.T. Rex, Thompson & Partners, Farnham, “Static analysis [software testing]”, IEEE, 19 Jun 1990, pp 4/1 - 4/3.
[9] Sheikh Umar Farooq, S. M. K. Quadri and Nesar Ahmad (2011), “Software Measurements And Metrics: Role In Effective Software Testing”, International Journal of Engineering Science and Technology (IJEST), Vol. 3 No. 1 Jan 2011.
[10] Hilda B. Klasky (2003), “A Study Of Software Metrics”, issued by Graduate School-New Brunswick Rutgers, The State University of New Jersey

AUTHORS PROFILE

Rohitt Sharma received his B.Tech degree in Computer Science from Lovely Professional University, India in 2011 and pursuing his M.Tech degree from Lovely Professional University, India. His current research interest includes Software Metric Performance Analysis, Mobile Operating Systems.

Paramjit Singh is Post Graduate in Computer Science and is currently working as Assistant Professor at Lovely Professional University. He has a teaching experience of around 5 years. He has 3 publications in his name including journal and conference

Sumit Sharma received his B.Tech degree in Computer Science from Lovely Professional University, India in 2011 and pursuing his M.Tech degree from Lovely Professional University, India. His current research interest includes Mobile Operating Systems, Software Metric Performance Analysis.