IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661, p- ISSN: 2278-8727Volume 16, Issue 2, Ver. VII (Mar-Apr. 2014), PP 64-69
www.iosrjournals.org

Measuring Maintainability Index of a Software Depending on Line of Code Only

Nahlah M.A.M.Najm
Al-Rafidain University College, Computer Engineering Technique Department

Abstract : *Maintainability Index (MI) has been proposed to calculate a single number that expresses the maintainability of a system. MI is calculated as a factored formula consisting of Lines of Code (LOC), Cyclomatic Complexity (CC)and Halstead Volume (HV). In this paper, we present a new method to find MI with respect to LOC only. To validate the method Measuring Maintainability Index Software (MMIS) is developed, that first finds MI with respect to (LOC, CC, and HV); secondly find MI with respect to LOC only. As the result has been compared they were closed but the new method was easy to understand, fast to count, and independent on the program language. In addition to many well-known programs, the MMIS was used to test itself and the result demonstrates that the program used to calculate MI value with respect to LOC only tend to be maintainable while the lexical analyzer the major step in calculating MI with regard to (LOC, HV, CC) is difficult to maintain.*

I. INTRODUCTION

The ISO/IEC 9126 standard [1] describes a model for software product quality that dissects the overall notion of quality into 6 main characteristics: functionality, reliability, usability, efficiency, maintainability, and portability. These characteristics are further subdivided into 27 sub characteristics. Furthermore, the standard provides a consensual inventory of metrics that can be used as indicators of these characteristics [2], [3].

Many software metrics have been proposed as indicators for software product quality in particular, Oman et al. proposed the Maintainability Index (MI) [9][22]. MI is a composite metric that incorporates a number of traditional source code metrics into a single number that indicates relative maintainability. The MI is comprised of weighted Halstead metrics(effort or volume) HV, McCabe's cyclomatic complexity (CC), Lines of codes(LOC)

As pointed by Van der Meulen & M.J.P Revilla, M.A[24], there are very strong connections between LOC and HV, LOC and CC. The study got some approximate expressions that have been used in this paper to formulate a new equation for MI that used in the developed Measuring Maintainability Index Software MMIS.MMIS assures the existence of following properties: to help reduce or reverse a system tendency toward "code entropy" or degraded integrity, to indicate when it becomes cheaper and/or less risky to rewrite the code than it is to change it, and fast convergence of MI value. This work presents a comprehensive, user friendly and general system based on LOC only, capable of dealing with any further program independent on the programming language.

II. LITERATURE SURVEY

Several maintainability models/methodologies were proposed to help the designers in calculating the maintainability of software so as to develop better and improved software systems. Starting from 1970s to 2012 various maintainability predicting models or techniques were developed. Bohem [4] presented quality model giving maintainability as one of its important attribute. Berns [5] concluded that maintenance factor depends on the level of difficulty to understand the software program. In 1985 Bowen put forward the equation to find out corrective maintainability. Sneed-Mercy Model[6], Robert Grady Model (at HP) [7], Geoferry and Kemerer Model [8].

Oman et.al [9]demonstrated that how software maintainability analysis can be used to guide software related decision making. Welker K. et.al [10] concluded that MI should not be interpreted in a vacuum rather it should be used as an indicator to direct human investigation. Muthanna et al. [11], developed a maintainability model using polynomial linear regressions. Polo et al. [25] used number of modification requests, mean effort per modification request and type of correction to examine maintainability. M. Dagpinar et .al [12] concluded that size and import direct coupling metrics are significant predictors for measuring maintainability of classes while inheritance, cohesion and indirect/export coupling measures are not. Hayes J.H et.al [13] maintainability model categorized software modules as ‘easy to maintain’ and ‘not easy to maintain’. The model helps the developers to identify the modules those are not easy to maintain, before integrating them. Van Koten[14] et.al BN based model to have better prediction accuracy than regression analysis based model for one out of two datasets i.e UIMS and QUES. Rizvi et.al [15] provided MEMOOD model giving improved maintainability or

understandability of class diagrams and Gautam C et.al [16] provided COMPOUND MEMOOD model which is much better than MEMOOD model giving not only understandability but modifiability, scalability and level of complexity of class diagrams that in turn leads to improved maintainability of software. Ruchika Malhotra et.al [17] estimated maintainability using machine learning algorithms and concluded that Group Method of Data Handling (GMDH) network model is one of the best modeling techniques to estimate maintainability of software. Alisara Hincheeranan et.al [18] calculated maintainability considering flexibility and extensibility as two sub characteristics of maintainability. Dubey et. al Model[19] used Multi Layer Perceptron (MLP) neural network model to predict maintainability using UIMS dataset and found model to be more accurate.

III. AIM OF THE RESEARCH

The aim of the research is suggesting a new equation to calculate maintainability index(MI) based on the line of code only, then verify the correctness of the suggested equation practically by designing software named Measuring Maintainability Index Software (MMIS).MMIS will compare MI value based on Halstead volume, line of code, cyclomatic complexity with the new ones using number of popular programs. The main steps of MMIS are explained in details furthermore the output tables of the presented software that will assure the accuracy and simplicity of the new equation also has been given.

IV. THE CLASSIFICATIONS OF THE METRICS OF SOFTWARE COMPLEXITY

The software metric is the measurement, usually using numerical rating, to quantify some characteristics or attributes of a software entity. Typical measurement includes the quality of the source codes, the development process and accomplished applications[20].

The metric of the software complexity is an essential and critical part of the software metric; it focuses on the quality of source codes.

Before discussing the details of the software complexity metrics, we classify the metrics in some ways [21].

Table 1. The Classification of the Complexity Metrics From The View of Software Lifetime

Metrics	Life time			
	Design	Code	complete	Maintain
McCabe	*	*		
Halstead			*	
Line of Code		*	*	
Error Count				*
Object Oriented Class Metrics	*	*		
Software Package Metrics	*		*	
Cohesion	*			
Coupling	*			

On the other hand, we can classify the metrics of software complexity by what they are calculated on. The classification of the 8 metrics sets is shown in table2.

Table 2. The Classification of the Complexity Metrics by Their Calculation Basis

Metrics	Target		
	Logic structure	Source Code	User Feedback
McCabe	*		
Halstead		*	
Line of Code		*	
Error Count			*
Object Oriented Class Metrics	*		
Software Package Metrics	*		
Cohesion	*		
Coupling	*		

V. MAINTAINABILITY INDEX (MI)

1. The old Equation

Maintenance is defined by the IEEE as "the process of modifying a software system or component after delivery to correct faults, improve performance or other attributes, or adapt to a changed environment" [22]. There have been several attempts to quantify the maintainability of a software system [25]. The most widely used software metric which quantifies the maintainability is known as Maintainability Index (MI).Maintainability Index is software metric which measures how maintainable (easy to support and change) the source code is. The maintainability index is calculated asafactored formula consisting of Lines of Code

(LOC), Cyclomatic Complexity (CC) and Halstead Volume (HV).According to Coleman, a MI value above 85 indicates that the software is highly maintainable, a value between 85 and 65 suggests moderate maintainability, and a value below 65 indicates that the system is difficult to maintain[22]. It is used in several automated software metric tools, including the Microsoft Visual Studio 2010 development environment, which uses a shifted scale (0 to 100) derivative.

Calculation

First we need to measure the following metrics from the source code:

- *HV* = Halstead Volume
- *CC* = Cyclomatic Complexity
- *LOC* = count of source Lines Of Code
- *CM* = percent of lines of Comment (optional)

From these measurements the MI can be calculated [22]:

The original formula:

MI = 171 - 5.2 * ln(HV) - 0.23 * (CC) - 16.2 * ln(LOC) .....(1)

The derivative used by SEI is calculated as follows [26]:

MI = 171 - 5.2 * log2 (HV) - 0.23 * CC - 16.2 * log2 (LOC) + 50 * sin (sqrt (2.4 * CM)) ..... (2)

The derivative used by Microsoft Visual Studio (since v2008) is calculated as follows [26]: MI = MAX (0, (171 - 5.2 * ln (Halstead Volume) - 0.23 * (Cyclomatic Complexity) - 16.2 * ln (Lines of Code))*100 / 171)

2. The Suggested Equation

In all derivatives of MI formula, the most major factor in MI is Line of Code which effectiveness has been subjected to debate.

By mean of research we found out that LOC has a number of important advantagessuch as:

a. Scope for Automation of Counting:

Since Line of Code is a physical entity; manual counting effort can be easily eliminated by automating the counting process. Small utilities may be developed for counting the LOC in a program.

However, a code counting utility developed for a specific language cannot be used for other languages due to the syntactical and structural differences among languages.

b. An Intuitive Metric:

Line of Code serves as an intuitive metric for measuring the size of software because it can be seen and the effect of it can be visualized. Function points are said to be more of an objective metric which cannot be imagined as being a physical entity, it exists only in the logical space. This way, LOC comes in handy to express the size of software among programmers with low levels of experience.

As a result we attempt to suggest anew equation that finds MI depending on LOC only. In order to derive the new equation we need to know that

HV=45*LOC-428 ..... (3) [24]

CC=0.22*LOC+1.9 ..... (4) [24]

By substituting the new HV, CC in equation (1) the original MI formula above we conclude the following new suggested MI equation

MI=171-5.2ln (HV (eq.3))-0.23*(CC (eq.4)-16.2ln (LOC) ..... (5)

VI. MMIS SYSTEM

Measuring Maintainability Index Software (MMIS) is software written in C++ language, developed to confirm the rightness of the new suggested equation (eq. 5). A number of known programs were given to the MMIS as an input then they were analyzed lexicallyto calculate Halstead Volume (HV),Line of Code (LOC),Cyclomatic Complexity (CC) which are needed to calculated MI value (according to eq.1) also, next subsection present the difference between the suggested equation complexity of calculating MI according to LOC only and calculating MI with respect to (HV, LOC, CC).Figure.1 shows the main structure of the designed MMIS system.

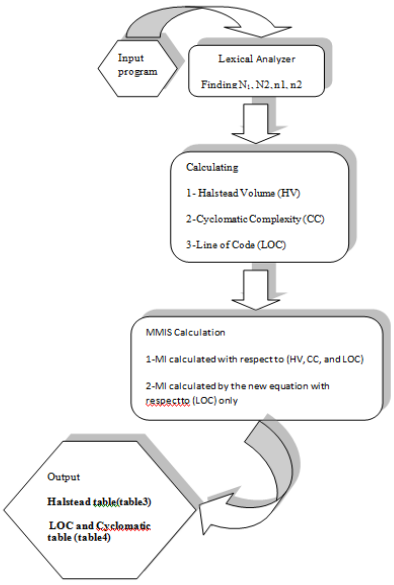

Figure 1. The MMIS System

MMIS Input

The input to the MMIS system is a program written in C++, with no restrictions on the program length. Merge, Quick sort, Merge sort, Insertion sort, lexical analyzer and LOC calculation program were the case studies that are used as an input to MMIS to assure the correctness of the new equation. These programs are given to the lexical analyzer procedure, the first step of the of the MMIS is:

Lexical Analyzer procedure

During this process, we may come across some identifiers, constants and keywords. These identifiers, constants and keywords are called tokens and can be stored in the symbol table as soon as we use or declare them for the first time and then these can be referenced later on. During lexical analysis such symbol table is built. Later these tokens can be classified as either operators or operands.

n_1 = number of distinct operators in the scanned program

n_2 = number of distinct operands in the scanned program

N_1 =number of operator occurrences

N_2 = number of operand occurrences

The values of n_1, n_2, N_1, and N2 will be considered as lexical analyzer output that are used in calculating Halstead's values in the next step.

Calculating Halstead's Values based on lexical analyzer output MMIS measures Halstead's Volume (number of bits required to specify a program), the program level (the measure of software complexity), the program difficulty, and other features such as development efforts and the projected number of faults in the software.

Program length (N) $=N_1+N_2$ ..... (6)[23]

Vocabulary size (n) $=n_1+n_2$ ..... (7)

Volume (V) $=N*\log_2 (n)$ ..... (8)

Difficulty level (D) $= (n_1/2)*(N_2/n_2)$ ..... (9)

Effort to implement (E) $=V*D$ ..... (10)

Time to implement (T) $=E/18$ ..... (11)

Calculating Line of Code (LOC) is the most traditional measure used to quantify software complexity. It is simple, easy to count, and very easy to understand. It does not, however, take into account the intelligence content and the layout of code. The Lines of Code count is usually for executable statements. It is actually a count of instructions statements.

Calculating Cyclomatic Complexity Metric (CC)

Measures the number of linearly-independent paths through a program module (Control Flow). The McCabe complexity is one of the more widely-accepted software metrics; it is intended to be independent of language and language format. In MMIS it is calculated as follows:

- Described (informally) as the number of simple decision points + 1
- Essentially the number of linearly independent paths through the code

- The code has no decision statements: complexity = 1
- The code has an if statement or switch or while or do, there are two paths through the code: complexity = 2

MMIS Calculation

Here the MMIS calculate MI with respect to the well known old equation with respect to Halstead Volume (HV),Cyclomatic Complexity (CC), and Line of Code (LOC) as presented in eq.1.Also MMIS calculates MI with respect to the suggested new equation which depends on LOC only as presented in eq.5. These calculations are accomplished to elucidate show the effect of applying the new equation.

MMIS Output Tables

These tables illustrate the efficiency and effectiveness of the new suggested equation by implementing both the old and the suggested MI equations on a number of popular programs as case studies.

The Halstead table will calculate Halstead Volume, Halstead difficulty, Halstead Effort Halstead Time. These values are necessary to determine the cases studies complexity further more Halstead Volume is needed along with the LOC, CC calculated in the second table to find MI with respect to the old equation (eq.1)

Finally maintainability table gives MI value for the tested case studies with respect to the old and new MI equations(eq.1 and eq.5 respectively) to determine if the tested programs were maintain or not. It is necessary to observe that the lexical analyzer, LOC calculating program were among the programs that has been given to the MMIS. The lexical analyzer is the main step for finding Halstead Volume needed in the old MI equation

Table (3) Halstead table

Program Name	V(Halstead Volume)	D(Halstead difficulty)	E(Halstead Effort)	T(Halstead Time to implement)
Merge	913.035	27	24651.945	1369.552
Quick sort	583.978	23	13431.494	746.194
Merge sort	416.167	26	10820.342	601.130
Insertion sort	280.460	21	5889.66	327.203
Lexical analyzer	3239	63	204057	11336.5
LOC calculation program	464.084	19	9132.510	507.36

Table (4) LOC and Cyclomatic table

Program name	LOC	CC
Merge	24	8
Quick sort	14	8
Merge sort	12	4
Insertion sort	6	3
Lexical analyzer	80	24
LOC calculation Program	16	5

Table (5) Maintainability table

Program name	MI (with old equation)	MI(with new suggested equation)	Maintainability
Merge	82.228	84.144	Moderate to maintain
Quick sort	93.284	99.4222	High to maintain
Merge sort	98.462	105.026	High to maintain
Insertion sort	111.974	141.232	High to maintain
Lexical analyzer	52.458	53.598	Difficult to maintain
LOC calculation Program	93.005	95.205	High to maintain

VII. CONCLUSION

In engineering, maintainability is the ease with which a product can be maintained in order to isolate defects or their cause, correct defects or their cause, repair or replace faulty or worn-out components without having to replace still-working parts, prevent unexpected value breakdowns, maximize a product's useful life, maximize efficiency, reliability, and safety, meet new requirements, make future maintenance easier, or cope with a changed environment.

Using the MI to assess source code and thereby identify and quantify maintainability is an effective approach. The MI provides one small perspective into thehighly complex issues of software maintenance. The MI provides an excellent guide to direct human investigation.

This paper provides some insight to the practical implementation of MI with a new,simple, effective method in comparison with the traditional method.

MMIS is software designed to assure the correctness of the new suggested equation as compared with the old one by testing it on number of case studies.

MMIS finds the complexity of any given program with respect to Halstead metrics, Line of Code, and cyclomatic complexity. MMIS proves through results that the new suggested equation can find an MIvalue which is very close to the value found by the old equation but with less time and effort accordingto the Halstead values.

MMIS approved the new suggested equation preciseness practically by measuring both complexity and maintainability of the lexical analyzer program used with the old MI calculating equation and LOC calculating program used with the new suggested equation.

MMIS Halstead output table; table (3), LOC table; table (4), and Cyclomatic output table; table (4) show that Lexical analyzer was much more complexity than LOC calculating program, also the maintainability output table, table (5), shows that LOC calculating program was high to maintain while lexical analyzer was difficult to maintain

References

[1] ISO, "ISO/IEC 9126-1: Software Engineering - Product Quality - part 1: Qualitymodel," Geneva, Switzerland, 2003.

[2] ISO, "ISO/IEC TR 9126-2: Software engineering - product quality -part 2: External metrics," Geneva, Switzerland, 2003.

[3] ISO, "ISO/IEC TR 9126-3: Software engineering - product quality -part 3: Internal metrics," Geneva, Switzerland, 2003.

[4] Boehm, B. W., J. R. Brown, H. Kaspar, M. Lipow, G. J. Macleod and M. J. Merit,1978, Characteristics of Software . Amsterdam: North Holland.

[5] G. M. Berns,"Assessing software maintainability", ACM Communications, 27(1), 1984.

[6] Sneed, H., Mercy, A. (1985), Automated Software Quality Assurance. IEEE Trans. Software Eng., 11Bi, 9: 909-916.

[7] Grady, Robert, Caswell, Deborah(1987), Software Metrics: Establishing a Company-wide Program. Prentic Hall. pp. p. 159.ISBN 0138218447.

[8] Gill Geoffrey K. and Chris F. Kemerer.(1991). "Cyclomatic Complexity Density and Software Maintenance Productivity,"IEEE Transactions on Software Engineering, Dec, pp.1284-1288.

[9] P. Oman and J. Hagemeister, "Metrics for assessing a software system's maintainability," Software Maintenance, 1992, pp. 337 - 344.

[10] Welker, K. and Oman, P.W., Software Maintainability Metrics Models in Practice, CrossTalk, Nov./Dec.1995, pp. 19-23 and 32

[11] S. Muthanna, K. Kontogiannis, K. Ponnambalaml and B. Stacey, "A Maintainability Model for Industrial Software Systems Using Design Level Metrics", In Working Conference on Reverse Engineering (WCRE'00), 2000

[12] M. Genero, M. Piattini, E. Manso, G. Cantone, "Building UML class diagram maintainability prediction models based on early metrics", Proceedings 5th International Workshop on Enterprise Networking and Computing in Healthcare Industry, , IEEE, 2003, pp. 263-275.

[13] Hayes, J. Huffman, Mohamed, N., Gao, T. The Observe-Mine-Adopt Model: An agile way to enhance software maintainability. Journal of Software Maintenance and Evolution: Research and Practice, Volume 15, Issue 5, Pages 297 – 323, October 2003.

[14] C.V. Koten, A.R. Gray, "An application of Bayesian network for predicting object-oriented software maintainability", Information and Software Technology Journal, vol: 48, no: 1, pp 59-67, Jan2006.

[15] Rizvi S.W.A. and Khan R.A. (2010) "Maintainability Estimation Model for Object-Oriented Software in Design Phase (MEMOOD)", Journal of Computing, Volume 2, Issue 4, April 2010.

[16] Gautam C, kang S.S (2011), "Comparison and Implementation of Compound MEMOOD MODEL and MEMOOD MODEL", International journal of computer science and information technologies, pp 2394-2398.

[17] Malhotra et.al," Software Maintainability Prediction using Machine Learning Algorithms." Software Engineering: An International Journal (SEIJ), Vol. 2, No. 2, SEPTEMBER 2012.

[18] Alisara Hincheeranan and Wanchai Rivepiboon,"A Maintainability Estimation Model and Tool." International Journal of Computer and Communication Engineering, Vol. 1, No. 2, July 2012.

[19] Dubey et.al."Maintainability Prediction of Object Oriented Software System by Using Artificial Neural Network Approach." International Journal of Soft Computing and Engineering (IJSCE) ISSN: 2231-2307, Volume-2, Issue-2, May 2012.

[20] Anthony M. Orme, Haining Yao, Letha H. Etzkorn, "Coupling Metrics for Ontology-Based Systems", IEEE Software, IEEE, Washington, March/April, 2006, pp. 102-108.

[21] Arpad Beszedes, Tamas Gergely, Szabolcs Farago, Tibor Gyimothy, Ferenc Fischer, "The Dynamic Function Coupling Metric and Its Use in Software Evolution", 11th European Conference on Software Maintenance and Reengineering, IEEE, Amsterdam, March 2007, pp. 103 - 112.

[22] D. M. Coleman, D. Ash, B. Lowther, and P. W. Oman, "Using Metrics to Evaluate Software System Maintainability." IEEE Computer, vol. 27, no. 8, pp. 44–49, 1994.

[23] R. E. Al-Qutaish and A. Abran, "An analysis of the design anddefinitions of Halstead's metrics," in 15th Int. Workshop on SoftwareMeasurement (IWSM'2005). Shaker-Verlag, 2005, pp. 337–352.

[24] van der Meulen, and M.J.P. Revilla, M.A, "Correlations between Internal Software Metrics and Software Dependability in a Large Population of Small C/C++ Programs", 18th IEEE International Symposium on Software Reliability, IEEE, Trollhattan, Nov. 2007, pp. 203 - 208.

[25] Anita Ganpati1, Dr. Arvind Kalia2, Dr. Hardeep Singh3," A Comparative Study of Maintainability Index of Open Source Software ",, International Journal of Software and Web Sciences 3(2), December, 2012-February, 2013, pp. 69-73.

[26] http://www.projectcodemeter.com/cost_estimation/help/Gl_maintainability.htm