Composite Complexity Measures (CCM)

P.S. GROVER^a AND N.S. GILL^b

^aDepartment of Computer Science, University of Delhi, India

^bDepartment of Computer Science & Applications, M. D. University Rohtak, Haryana, India

Abstract

Studies based on Halstead and McCabe's metrics use different attributes of a program to measure the software complexity. There are certain characteristics which are used by software science measures but the same are not used by cyclomatic number measure to ascertain the complexity and vice-a-versa. Here, we report the results of our study which uses the combined characteristics of the program to measure the software complexity. This has been done by employing the concept of weights for the various tokens occurring in control/iterative structures in a program.

Keyword Codes: D.2.8; D.4.8

Keywords: Complexity Measures, Software science, Cyclomatic number; Development effort; Composite measure.

1. INTRODUCTION

The complexity of software systems affects its design, testability, reliability and maintainability. In order to produce reliable software, its complexity must be controlled by suitably decomposing the software systems into smaller subsystems [1]. The ever increasing cost of program development has made the measurement of software complexity much more important than it has ever been before [2]. There are several ways of perceiving the software systems for the purpose of complexity measurement. Out of these software science complexity measures and the cyclomatic complexity measure are prominent ones [3,4]. Though both these methods measure software complexity, but basis for measuring software complexity is completely different.

In software science measure, the basis of detecting and measuring the software complexity is the size of software systems only and it doesn't keep track of other characteristics of software systems such as program flow control, iteration and so on [3,5]. Similarly, cyclomatic number measures the software complexity due to program flow control only, but doesn't keep track of other characteristics of software systems such as operators, operands, etc. In this paper we have attempted to take into account the complexities both due to size of software systems as well as due to presence of various control/iterative structures in modules of the software systems.

Recently, Bina Ramamurthy and Austin Melton [6] have tried to measure the weighted complexity of software modules by assigning additional weights to total operator and operand counts. But they did not take into account the multiple-test predicates, GOTO statements, etc. In present study, we have examined these issues as well. We have used our three-term model [7] and taken into account these various multiple-test predicates. We call this a Composite Model. The measures will be referred to as *Composite Complexity Measure*.

Before defining our composite measure, we briefly introduce software science measures (Halstead as well as our three-term model) and McCabe's cyclomatic complexity measure [3,7,4].

2. HALSTEAD's SOFTWARE SCIENCE MEASURE

In software science [3] approach,
vocabulary size is given as : $n = n_1 + n_2$

where n_1 , n_2 are number of unique operators and operands respectively.

The length of program is $N = N_1 + N_2$, where N_1 and N_2 are total occurrences of operators and operands respectively. Other metrics such as program volume (V), language level (L), programming effort and time etc. are defined using these basic metrics.

As the modern languages are very rich in functions/reserve-words/keywords so it is better to count keywords/reserve-words/functions separately besides operators and operands [7]. Using this model, the results for estimating program lengths have also been found encouraging. We apply this model to our present study.

Here n_1 , n_2 are the unique number of operators, operands respectively whereas n_3 is the number of unique keywords/ reserve-words/ functions which was earlier considered a part of operators. Similarly, N_1 , N_2 , N_3 are the total count of n_1 , n_2 , n_3 respectively.

Following Halstead measures, vocabulary size is defined as : $n = n_1 + n_2 + n_3$ and the length of program is $N = N_1 + N_2 + N_3$.

These are now used to estimate other measures such as program volume, language level, effort and so on.

3. McCabe's CYCLOMATIC COMPLEXITY MEASURE

M McCabe measure [4] is based upon the study of program control graphs. If G is the program control graph for a given program, then the cyclomatic number is : $V(G) = mn + p$, where m is number of arcs, n is number of nodes, p is number of connected components ($p=1$ for strongly connected control graphs). McCabe's measure has been reasonable for control cyclomatic complexity but doesn't take into account the software complexity due to size as mentioned earlier.

4. COMPOSITE COMPLEXITY MEASURES

Composite measure takes into account both the complexities due to software size as well as control flow. It has been done by associating extra weights to the total occurrences of software tokens in our model. We assign weights to the nonsequential structures so that length, volume, effort, language level, programming time etc. can detect complexities due to nonsequential structures [6,8,9].

Rules for assigning extra weights and levels are as follows :

- i) If there is no non-sequential structure present in the software module, then extra weights will be zero for all occurrences of operators,operand,keywords/reserve-words/functions.
- ii) List of the nonsequential structures/predicates irrespective of the level of control structure with their corresponding weights are :

<i>STRUCTURE /Predicates</i>	<i>Weights</i>
a) if-then/if-then-else/ repeat-until/do-while/ for-do	1
b) GOTO/Computed GOTO (or CASE)	1
c) AND/OR/EQV/XOR	2
- iii) Level of the control structures will be increased by one for a single occurrence of any control structure within its body (nesting) and so on.

iv) Only the control structures will be examined for assigning weights and levels to all the operators, operands and keywords/reserve-words/functions respectively so that additional values due to weight and level could be added to the present values of N_1 , N_2 , N_3 respectively.

v) Nesting of heterogeneous (different types) control structures will also be considered in the same fashion as nesting of homogeneous control structures.

vi) The control structures such as if-then-else-if will be considered as a multilevel nesting and will be converted into simple structures using the control structure if-then-else for the purpose of assigning weights.

Following the above mentioned rules for assigning weights and levels, the weighted metrics are given as :

$$N_w = N_{w_1} + N_{w_2} + N_{w_3}$$

where N_{w_1} , N_{w_2} , N_{w_3} are weighted total count of n_1 , n_2 , n_3 respectively and are defined as :

$$N_{w_1} = N_1 + \int_{n_1} DX(n_1) * L(n_1)$$

$$N_{w_2} = N_2 + \int_{n_2} DX(n_2) * L(n_2)$$

$$N_{w_3} = N_3 + \int_{n_3} DX(n_3) * L(n_3)$$

where $DX(t)$ is the weight attached to token t and $L(t)$ is the level at which the token t is positioned.

Weighted volume, $V_w = N_w * \log_2 n$, Weighted effort, $E_w = V_w / \hat{L}$

Weighted Language level, $l_w = L_w * V_w^*$, Weighted time, $T_w = E_w / 18$.

We have applied these rules to various languages like FORTRAN, Pascal, C and BASIC. But, here we report our results for FORTRAN 77 only.

5. EXPERIMENTAL MENTAL RESULTS

We take two test programs P1 and P2 listed below :

```
C   Test program P1
    REAL U, V, W, X
    READ(*,*)U,V,W
    X=U
    U=V
    W=X
    X=U+V+W
    WRITE(*,*)X
    END
```

```
C   Test program P2
    REAL U, V, W, X
    X=U
    IF (X.LT. V) X=V
    IF (X.LT. W) X=W
    WRITE(*,*)X
    END
```

Applying theory of software science with revised basic metrics [7], we get the following results :

For P1

For P2

$n_1=3, n_2=4, n_3=3; N_1=9, N_2=16, N_3=3$ $n_1=3, n_2=4, n_3=4; N_1=7, N_2=14, N_3=5$
 $n = 3+4+3 = 10; N = 9+16+3 = 28$ $n = 3+4+4 = 11; N = 7+14+5 = 26$
 $V = 93, l = .67, E = 1116, T = 62$ $V = 90, l = .57, E = 1103, T = 61$

By looking at the values of different metrics, it seems that both the programs P1 and P2 need the same programming effort and time but by looking at the programs, program P1 is found less complex than program P2. In other words, this study doesn't reflect the effect of control structures.

Applying McCabe's theory of complexity for the same programs P1 and P2 the results are :

$$V(G1) = m - n + 1 = 1; \quad V(G2) = m - n + 1 = 3.$$

Thus by looking at the cyclomatic numbers, we infer that program P2 is more complex than program P1. But, this study does not take care of the complexity which will arise due to size of software modules.

Now, by applying the composite measures, we get the following weighted results :

For P1:

$$Nw_1 = N_1 + 0 = 9; \quad Nw_2 = N_2 + 0 = 16;$$

$$Nw_3 = N_3 + 0 = 3; \quad n = 3 + 4 + 3 = 10$$

$$Nw = 9 + 16 + 3 = 28; \quad Vw = 93$$

$$Ew = 1116, \quad Tw = 62, \quad lw = .67$$

For P2:

$$Nw_1 = N_1 + ((1+1)) = 9; \quad Nw_2 = N_2 + (1+1+(1+1)) = 18$$

$$Nw_3 = N_3 + ((1+1)) = 7; \quad n = 3 + 4 + 4 = 11$$

$$Nw = 9 + 18 + 7 = 34; \quad Vw = 118$$

$$Ew = 1858, \quad Tw = 103, \quad lw = .44$$

From these calculations we infer that program P1 is less complex than program P2. Differences arise due to two factors : size and control/iterative structures.

We have studied a set of thirty two FORTRAN 77 language programs for all the three measures. Following the weighted results we have computed program volume, language level, development effort and time which are given in Table I along with the values of lines of code (LOC) and McCabe's cyclomatic number, V(G). We have also estimated the relative differences between composite metric Ew and E as a function of McCabe cyclomatic number, V(G) and has been plotted in Fig. 1. We observe that with the rise in McCabe's complexity the effort increases.

The results of composite measures can be used for modularizing the software systems rather than using both software science metric E and McCabe's cyclomatic number V(G).

6. CONCLUSIONS

In the present study we have applied the three term model taking into account the effect of control/repetitive structures on the software science metrics. Our study shows that the effect of these structures on complexity is quite significant.

ACKNOWLEDGMENTS

We would like to thank Prof. D.R. Taneja of M.D. University, Rohtak for the many useful suggestions and kind help he has given during the preparation of this paper.

REFERENCES

1. Lew, Ken S., Dillon, T.S. and Forward, K.E. : 'Software Complexity and Its impact on Software Reliability', IEEE Trans. on SW Engg., November 1988, Vol. 14, (11).
2. Shen, Conte, and Dunsmore : 'Software Science Revisited : A critical analysis of the theory and its empirical support', IEEE Trans. on SW Engg, March 1983, Vol. SE-9,(2), pp. 156-165.
3. Halstead, M.H. : 'Elements of Software Science', (New York, Elsevier, North-Holland, 1977).
4. McCabe, T.J. : 'A Complexity Measure', IEEE Trans. on SW Engg., December 1976 Vol. SE-2, (4), pp. 308-320.
5. Conte, S.D. and Shen, Dunsmore : 'Software Engineering Metrics and Models', (The Benjamin/Cummings Pub. Company, Inc.. 1986).
6. Ramamurthy, Bina and Melton, Austin : 'A Synthesis of Software Science Measures and the Cyclomatic Number', IEEE Trans. on SW Engg., August 1988, Vol. 14, (8), pp. 1116-1121.
7. Grover, P.S., Gill, N.S. and Singh, R. : 'Measuring Software Systems', Proc. Annual National Seminar on "Operations Research and Management Decision Making", New Delhi, Nov. 19-20, 1993.
8. Prather, Ronald E. : 'An Axiomatic Theory of Software Complexity Measure', The Computer Journal, 1984, Vol. 27, (4).
9. Harrison, W. and Magel, K. : 'A Complexity Measure Based On Nesting Level', SIGPLAN , March 1981, 16(3).

Table - I
Composite Metrics for FORTRAN ?? Language

P.No.	V	Vv	LOC	λ	λv	V(G)	E	Ew	T	Tv
1	904	1067	15	2.07	1.75	4	18794	26216	1044	1456
2	443	568	10	1.31	1.09	5	8099	13028	450	724
3	606	691	14	3.21	2.83	5	8288	10747	460	597
4	198	250	9	0.7	0.5	3	3379	5482	188	306
5	495	807	17	0.41	0.28	7	16952	43859	942	2437
6	781	1177	20	0.47	0.34	10	31748	69645	1764	3869
7	803	960	17	0.79	0.69	6	25331	36090	1407	2005
8	1489	2598	51	0.35	0.21	11	97320	288667	5407	16037
9	721	1178	28	0.74	0.49	9	22322	57745	1240	3208
10	1900	2519	50	0.57	0.44	11	109195	190833	5066	10602
11	348	719	15	0.42	0.2	6	10029	42544	557	2364
12	768	902	26	0.47	0.44	5	30968	41800	1720	2278
13	835	1184	33	0.64	0.48	7	30036	59497	1659	3305
14	1472	1611	20	0.89	0.81	5	59595	71283	3311	3960
15	644	1199	29	0.19	0.1	9	37225	128925	2060	7183
16	664	1136	17	0.54	0.32	7	23380	87964	1299	3776
17	564	636	19	0.97	0.9	3	13097	16916	728	940
18	1174	1638	23	1.16	0.8	7	37508	73784	2084	4099
19	1191	1850	29	0.86	0.56	9	44440	105714	2489	5873
20	3732	4685	50	2.04	1.65	10	159487	250635	8860	13919
21	964	1183	13	0.47	0.38	5	42780	65722	2377	3651
22	502	701	17	0.34	0.28	4	18943	35684	1052	1977
23	1105	1432	29	0.41	0.32	9	57254	95467	3181	5304
24	1768	2542	41	0.54	0.38	11	102197	206667	5678	11482
25	263	291	11	2.18	1.98	2	2893	3523	161	196
26	260	520	12	0.55	0.28	5	5652	22222	314	1235
27	201	309	7	0.33	0.25	4	4926	11196	274	622
28	53	53	3	1.54	1.54	1	309	309	17	17
29	405	590	16	0.55	0.42	5	11066	22433	615	1246
30	619	855	20	0.53	0.42	5	20983	38514	1166	2140
31	93	93	7	0.67	0.67	1	1116	1116	62	62
32	90	118	5	0.57	0.44	3	1103	1858	61	103

Fig. 1 : Plot Between V(G) and (Ew-E)