A NOVEL APPROACH FOR COMPLEXITY MEASURE
BASED
ON CONTROL STRUCTURES

Aparna Shukla
Department of Computer Science
Birla Institute of Technology, Mesra, EC Naini
Allahabad, UP-India
E-mail: meet2aparna@gmail.com

Dr. Prabhat Ranjan
Department of Computer Science
Motilal Nehru National Institute of Technology
Allahabad, UP-India
E-mail: prabhat@mnnit.ac.in

Abstract

Metrics plays an important role in the software modularization. Several metrics have been discussed so far. This paper describes a metrics based on control structures which is a minor modification of McCabe’s approach. Proposed novel approach is a control structure based complexity (CSBC) which is used to compute the complexity by counting the control structures directly from the source code rather than finding out predicate node from the control flow graph (CFG) as in McCabe cyclomatic complexity metric. Proposed novel approach also overcomes the potential implementation problem of the existing McCabe approach based on predicate node that only applies to individual module (function) rather than on n modules. To effectively aid the software assessment a metric tool is developed to support the software metric. Therefore, this paper introduced a software metric tool (SMT) to provide an automated software metric support for users.

Keywords- Cyclomatic complexity measure; Control structures; Predicate node.

I. INTRODUCTION

IEEE Standard Glossary of Software Engineering Terms define metric as “a quantitative measure of the degree to which a system, component, or process possesses a given attribute”.

One important critical issues faces by software engineering is the modularization of a software system so that the modules are both testable and maintainable. A mathematical technique is needed to provide a quantitative basis for modularization of the system, so that the resulting modules that are difficult to test or maintain can be identified [1].

Several different approaches has been suggested to measure the software complexity such as McCabe Cyclomatic complexity[1], Halstead programming effort [2], Chidamber’s metric for object oriented[3], Klemola’s KLCID complexity measure[4], Wang’s cognitive functional complexity[5], Kushwaha’s cognitive information complexity measure[6] and many more.

All the existing complexity measures are supposed to cover the correctness, effectiveness and clarity of software and to provide a good estimate of cost of testing, maintenance, number of errors, cost, development time, efforts, etc. Out of the several proposed measures which complexity measure should be chosen for software is again a problem, as every measure has its own pros & cons? There is a continuous effort to find such a comprehensive complexity measure, which addresses most of the parameters of software.
This paper is organized as follows. First, in Section II a complexity measure is explained together with the McCabe cyclomatic complexity and gives an intuitive explanation of this metric with examples. Next, in Section III paper explain the concept of the proposed novel approach. This is followed in Section IV by an example to validate the proposed novel approach. Section V gives a comparative study of proposed control based complexity measures with existing classical McCabe’s approach. Section VI discuss the SMT an automated tool which is built in & application in complexity computing of C code briefly and lastly, in Section VII conclusion of the paper with some comments are made about future research directions.

II. COMPLEXITY MEASURE

Complexity in general is defined as "the degree to which a system or component has a design implementation that is difficult to understand and verify ". Complexity metrics are used to predict critical information about reliability and maintainability of software systems [9].

In 1976, McCabe was the first to put forward the Cyclomatic Complexity, one of the most widely accepted metric. This metric is an independent of the language as well as language format. It is a software measure which is used to define the complexity of a program. Control flow graph (CFG) of the program is used to compute this metric. The points (node) of the CFG correspond to the command statement of the program and directed edge connects two nodes represents flow of command statement in sequential manner.

McCabe [1] developed a mathematical technique computed using directed graph to describe the control flow of the program. This measure is based on concept of graph theory and shows that the complexity is independent of physical size and depends only on the decision structure of a program.

Following theorem and corollary are used to compute complexity [8]:

**Theorem 1:** The cyclomatic number \( V(G) \) of a graph \( G \) with \( n \) nodes, and \( e \) edges is

\[
V(G) = e - n + 2 \tag{1}
\]

**Corollary 1:** The cyclomatic number \( V(G) \) of a graph \( G \) can be computed using the predicate node \( p \) (a node having two outgoing edges) is

\[
V(G) = p + 1 \tag{2}
\]

**Corollary 2:** The cyclomatic number \( V(G) \) of a graph \( G \) can also be computed using the number of regions \( r \) found is

\[
V(G) = r + 1 \tag{3}
\]

A computer program associated with a directed graph that has unique entry and exit nodes. In the graph each node corresponds to a set of instruction in the program where the flow is sequential, and the edges correspond to the branches taken in the program. The graph is known as the program control graph, and the cyclomatic number is then a complexity measure for the program. This approach suggests that complexity is increased by branches and can be reduced by removing unnecessary branches. The complexity computation can be done using any one of the equation prescribed above.

Let consider an example to find the given number is even or odd as shown below-

**Example #1:**
Upon going through given code and its CFG above, the described algorithms are applied and the following parameters can be extracted as:

By using Theorem 1, where \( e=7, n=7 \), then
\[
V(G) = 7-7+2 = 2
\]

By using Corollary 1, where \( p=1 \), then
\[
V(G) = 1+1 = 2
\]

By using Corollary 2, where \( r=1 \), then
\[
V(G) = 1+1 = 2
\]

In general, a procedural code segment consists of one or more control structures containing related set of statements. The cyclomatic complexity metrics developed for procedural language are mainly based on calculating the number of edges, number of nodes, number of predicate nodes etc.

However, the complexity of an entire application is more important than the complexity of an individual module. Therefore, it is required to compute the complexity of an individual module as well as an entire application.

The corollary 1 has some implementation problem associated with it that applies only to the individual modules [7]. As mentioned earlier that developer may be interested to compute the complexity for an entire application consist of \( n \) modules.

There are two possible ways to compute the complexity of an overall application. The first possible way to apply the formula on individual module and summed the results of an all individual to determine the cyclomatic complexity for an entire application. The second possible way to sum the predicate node of all the modules first and then apply the formula, it leads towards the incorrect result as both the result is differing.

Let consider an example to find the factorial of the given number as shown below to illustrate this problem-

Example #2:

```c
void main ( )
{
    int num, factorial;
    clrscr ( );
    printf ("Enter number to calculate the factorial");
    scanf ("%d", &num);
    factorial=fact (num);
    printf("Factorial of %d is %d", num, factorial);
}
```

Fig.1 CFG of main function
printf ("Factorial=%d", factorial);
getch ();
}

int fact (int n)
{
    if (n==0)
        return 1;
    else
        return n*fact (n-1);
}

The CFG for the above code is as below-

![CFG Diagram](image)

Let compute the cyclomatic complexity for Fig. 2 by using corollary 1.

**Case 1:** Apply formula to individual module and then summed the individual results to obtain the complexity of an entire application-

Complexity for Fig. 2(a) where \( p_a = 0 \), then

\[
V_a (G) = 0 + 1 = 1
\]

Similarly, Complexity for Fig. 2(b) where \( p_b = 1 \), then

\[
V_b (G) = 1 + 1 = 2
\]

where \( V_a (G) \), \( V_b (G) \) is a cyclomatic complexity and \( p_a, p_b \) are the predicate node for the Fig. 2(a) & 2(b) respectively.

Now the Cyclomatic Complexity of an entire application is

\[
V (G) = V_a (G) + V_b (G)
\]

\[
V (G) = 1 + 2 = 3
\]

**Case 2:** Adding the predicate node of all modules and then apply formula to obtain the cyclomatic complexity of an entire application-
For Fig. 2(a): \( p_a = 0 \)

For Fig. 2(b): \( p_b = 1 \)

Total number of predicate node in an entire application is \( p = p_a + p_b = 0 + 1 = 1 \)

Now applying formula

\[ V(G) = p + 1 = 1 + 1 = 2 \]  \[(5)\]

By the (4) & (5) of case 1 and case 2, it is clear that case 2 underestimate the result. Therefore, it is true that the traditional approach applies to only individual module.

The problem associated with the exiting corollary 1 given by McCabe is overcome by a proposed novel approach given in this paper.

III. PROPOSED NOVEL APPROACH

Proposed novel approach is used to compute the complexity based on counting the control structures directly from the source code rather than counting the predicate node from the control flow graph (CFG) as in McCabe cyclomatic complexity metric.

Proposed novel approach also overcome the limitation of the existing McCabe approach based on predicate node that only applies to individual module rather than a application consist of \( n \) modules.

The control based complexity measure of an entire software CSBC is defined as the sum of control based complexity of its \( n \) individual module. Let there are \( n \) modules in the given software, then CSBC can be calculated as

\[ \text{CSBC} = \sum_{i=1}^{n} \text{Number of CS} + n \]  \[(6)\]

where CS stands for Control structures (defined in ANSI C) and \( i \) represents number of module and \( n \) is the total number of module.

The following points are to be considered for counting the control structure while using proposed method (CSBC) to compute complexity:

- Every control structure counted separately as they contribute a fixed amount of complexity.
- Boolean operators are also counted individually if they are used within control structures.
- If multiway decision are constructs such as switch statement with several cases. Then all “case labeled statements” are counted individually only.

IV. WORKING OF PROPOSED NOVEL APPROACH

This section detailed the basic steps of the proposed novel approach and validates it to measure the complexity given in (6) with the help of examples.

Basic Working

1. Accept source code
2. Count the total number of control structure in the code.
3. Count the total number of module in the code.
4. Apply formula derived in (6).

Now let’s consider two example to validate the proposed approach, first that contains one module and second example contains two modules.

Firstly considering an example to find whether the given number is prime or not as shown below-

Example #3:

```java
boolean isprime = true;
int i = 2;
while (i<n) {
    if (n%i == 0) {
        isprime = false;
    }
    i++;
}
print (isprime);
```

Upon going through code above the following parameters can be extracted-

Total number of module (n) =1  
Number of CS=2  
Therefore, by using (6) the CSBC can be calculated as  
CSBC=2+1=3  
Thus, the CSBC value is 3.

Let consider another example that calculate the average of a set of numbers as shown below-

Example #4:

```java
# define N 10
main () {
    int count;
    float sum, average, number;
    sum=0;
    count=0;
    while (count<N) {
        scanf (“%f”, &number);
        sum = sum + number;
        count = count +1;
    }
    average= sum/N;
    printf (“N=%dsum = %f”, N, sum);
    printf (“average = %f”, average),
}
```

Upon going through code above the following parameters can be extracted-
Total number of module (n) = 1
Number of CS = 1

Therefore, by using (6) the CSBC can be calculated as

$$\text{CSBC} = 1 + 1 = 2$$

Thus, the CSBC value is 2.

Let consider another program (contain two modules) to be developed for printing the Fibonacci series up to n terms as shown below-

Example #5:

In the given C code, the parameters are

Number of modules (n) = 2
Number of CS in module 1 (main ( )) = 0
Number of CS in module 2 (fact ( )) = 1

Therefore, by using (6) the CSBC can be calculated as

$$\text{CSBC} = (0 + 1) + 2 = 3$$

Thus, the CSBC value is 3.

Hence, from the examples illustrated above it is clear that the proposed novel approach is capable to calculate the complexity of the program that contain individual module as well as the program that consist more than one module.
V. RESULT AND COMPARISON

This segment of the paper analyzes the result of applying CSBC on the program given below and find that the lacunae in the traditional approach McCabe’s corollary 1 has been successfully dealt with in the approach given in this paper.

Let consider an example shown below and comparing the result of both the traditional McCabe approach and the proposed novel approach.

Example #6:

```c
void main()
{
    int a, b, ch;
    clrscr();
    printf("Enter the two values:");
    scanf("%d%d", &a, &b);
    add(a, b);
    subtract(a, b);
    multiply(a, b);
    division(a, b);
    getch();
}

void add(int a, int b)
{
    int result;
    result = a + b;
    printf("result=%d", result);
}

void subtract(int a, int b)
{
    int result;
    if (a > b)
        result = a - b;
    else
        result = b - a;
    printf("result=%d", result);
}

void multiply(int a, int b)
{
    int result;
    result = a * b;
    printf("result=%d", result);
}

void division(int a, int b)
{
    int result;
    if (a > b)
        result = a / b;
    else
```
result = b/a;
printf("result=%d",result);
}

The CFG for above C code is as below:

(a)  

(b)  

(c)  

(d)  

(e)  

Fig. 3(a) CFG of main () 3(b) CFG of add () 3(c) CFG of subtrate ()
3(d) CFG of multiply () 3(e) CFG of division ()

Now calculation of measures by McCabe’s corollary 1 and proposed novel approach CSBC are as-

**Case 1:** Apply formula to individual module and then summed the individual results to obtain the complexity of an entire application
TABLE I.  COMPARISON OF RESULT IN CASE 1

<table>
<thead>
<tr>
<th>Module</th>
<th>By McCabe’s corollary 1</th>
<th>By Proposed novelapproach</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$V(G) = p+1$</td>
<td></td>
</tr>
<tr>
<td>main()</td>
<td>$p=0$</td>
<td>$V_{max}(G)=1$</td>
</tr>
<tr>
<td>add()</td>
<td>$p=0$</td>
<td>$V_{add}(G)=1$</td>
</tr>
<tr>
<td>subtract()</td>
<td>$p=1$</td>
<td>$V_{subtract}(G)=2$</td>
</tr>
<tr>
<td>multiply()</td>
<td>$p=0$</td>
<td>$V_{multiply}(G)=1$</td>
</tr>
<tr>
<td>division()</td>
<td>$p=1$</td>
<td>$V_{division}(G)=2$</td>
</tr>
<tr>
<td>Summing the result of all individual modules</td>
<td>$V(G)=1+1+2+1+2=7$</td>
<td>$CSBC=1+1+2+1+2=7$</td>
</tr>
</tbody>
</table>

**Case 2:** Counting the number of CS of all modules and then apply formula to obtain the complexity of an entire application.

TABLE II.  COMPARISON OF RESULT IN CASE 2

<table>
<thead>
<tr>
<th>Module</th>
<th>By McCabe’s corollary 1</th>
<th>By Proposed novelapproach</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$V(G) = p+1$</td>
<td></td>
</tr>
<tr>
<td>main()</td>
<td>$p=0$</td>
<td>$CS_{main}=0$</td>
</tr>
<tr>
<td>add()</td>
<td>$p=0$</td>
<td>$CS_{add}=0$</td>
</tr>
<tr>
<td>subtract()</td>
<td>$p=1$</td>
<td>$CS_{subtract}=1$</td>
</tr>
<tr>
<td>multiply()</td>
<td>$p=0$</td>
<td>$CS_{multiply}=0$</td>
</tr>
<tr>
<td>division()</td>
<td>$p=1$</td>
<td>$CS_{divide}=1$</td>
</tr>
<tr>
<td>Summing predicate node of all individual modules</td>
<td>$V(G)=(0+0+1+0+1)+1=3$</td>
<td>$n=5$ $CSBC=(0+0+1+0+1)+5=7$</td>
</tr>
</tbody>
</table>

From Table I and Table II, it is clear that proposed novel approach in this paper gives the same result in both cases unlike to traditional approach that gives different result for multiples modules.

VI.  SOFTWARE METRIC TOOL

The software metric tool (SMT) has been developed by using C# language under user friendly .Net 2005 framework environment. This section also describes the working of the developed tool.

It is tedious job to find the complexity of the program manually, so by the use of metric tool it become easier. The Tool described in this paper is simple and used to compute complexity of programs by some popular classical existing method as well as proposed novel approach. This is assumed that all metrics tools compute / interpret / implement the same metrics in the same way.

Software metrics tool can be defined as a program, this implement a set of software metrics definitions. It allows to assess a software system according to the metrics by extracting the required entities from the software and providing the corresponding metrics values.

According to the specified model, the tool calculates the related program information and display the result of the input code. Fig. 4 shows the architecture of SMT.
In the analyzing phase, the front end analyzes the source code, extract the program information. In the calculating phase, according to the selected metric models which have already been customized by the user, are calculated and finally the metric result is displayed on the interface.

In general the proposed software metric tools calculate different metrics values for the same input. It analyzes the C program complexity of metric including classical approach such as LOC and proposed metric method.

The Fig. 5 shows the main frame of the SMT. The main feature of this form is activated by first selecting the file to and the respective model which the user wants to calculate the complexity.

The Fig. 6 shows the result of one of the classical software metric based on length of the source code i.e. Line of Code (LOC).
The Fig. 7 shows the result of software science based metric which consider the basic elements of code, operator and operand.

![Fig. 7 Result of Halstead](image)

The Fig. 8 shows the result of McCabe cyclomatic complexity based on a graph-theoretic approach and also gives the corresponding control flow graph of the considered C program.

![Fig. 8 Result of McCabe Cyclomatic Complexity](image)

The Fig. 9 shows the result of proposed novel approach CSBC, which can be computed directly from the source code.

![Fig. 9 Result of CSBC](image)

The Fig. 10 shows the comparison result of McCabe cyclomatic complexity with the proposed approach.

![Fig. 10 Comparison Result](image)
VII. CONCLUSION

In this paper a novel approach for complexity measure has been presented, which is a slight modification of the McCabe’s approach and based on counting the control structure. It is a simplest and straightforward method because it can be measured directly by the given code. Further, it also solves the potential implementation problem of the McCabe’s cyclomatic complexity $V(G) = p+1$. The software tool introduced in this paper compute the complexity of C programs by using some classical approach as well as proposed novel approach given in this paper. SMT is designed and developed to support code based software assessment.

Future research is required to extend more complexity factor as well as apply to other application areas also such as network, communication and many more. Additionally, the tool will be enhanced so that it is compatible with other languages too.

REFERENCES


