Volume 5, Issue 7, July 2015 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Approaches to Improve Code Complexity Analysis

Abhilasha K, Harshitha B S, Bhavya P, Pavithra P, Dr.Saroja Devi H
Department of CSE, NMIT,
India

Abstract: Complexity is always considered as an undesired property in software since it is a fundamental reason for decreasing software quality and is indicative of defects. A number of metrics are proposed for calculating the code complexity which includes both quantitative and qualitative approach. Cyclomatic Complexity(CC) is the most widely used complexity metric based on McCabe's approach. The complexity calculated using the existing approaches including CC does not achieve the objective to measure the complexity of the entire system in design phase.Cyclomaticcomplexity defined by McCabe computes a complexity value within a procedure and it does not take into account the interaction between the different modules along with interaction within the modules. In this paper we present approaches to implement two new formulas to measure the system's complexity, viz., Total Cyclomatic Complexity (TCC) and Coupled Cyclomatic Complexity(CCC) in the design phase.TCCexplains the inter-modular complexity i.e., interactions between different modules and CCC explains the intra-modular complexityin a system i.e., interactions within the same module.

Keywords: Cyclomatic Complexity, Total Cyclomatic Complexity, Coupled Cyclomatic Complexity, Software quality

I. INTRODUCTION

Complexity[6] is an important factor that defines the quality of software from the point of reusability, ability to understand, and cost of maintenance[2]. Higher degrees of complexity is usually an undesired property of software because complexity makes software harder to read and understand, and therefore harder to change. Complexity is believed to be one cause of the presence of defects, leading to consider that software complexity is the contrary of software quality.Complexity analysiscan be incorporated intovarious stages of the software life cycle: requirement analysis, design, implementation and testing in order to improve the overall quality and reliability. Complexity measures help reduce the time-to-market and win larger market segments because of the resulting improvement in addressing the criticality and rapid development features.

II. RELATED WORK

Several software works activities have focused on developing complexity metrics, to measure the software quality in order to improve it and predict its faults. The two types of complexity metrics are quantitavie and structural metrics.

- **Quantitative metrics:**The quantitative metrics are those that take into account the quantitative characteristics of code. Generally it measures size which is supposed to be proportional to many factors in software development including effort and complexity.
- **Structural Metrics:**These are pertaining to the design structure of the program.

Structural metrics can be classified into two main different categories, which are explained below:

1) Control-Flow Metrics:The Control Flow structure is concerned with the sequence in which instructions are executed in a program.
2) Data-Flow Metrics Inter-Modular: When the measures of the control-flow structure focus on the attributes of individual modules, the measures of data-flow structure emphasize the connections between the modules and these metrics are called Inter Modular or data flow metrics.

- **Cyclomatic Complexity:**

The cyclomaticcomplexity[4]of a section of source program is the count of the number of linearly independent paths in the source code. Defined by Thomas McCabe[3]in 1976itgivesstructural complexity[4] of a procedure based on the control flow.It is easy to understand, easy to calculate and it gives useful results. The procedure's statements are first transformed into a graph. Then the cyclomatic complexity is calculated by measuring the linearly independent path in the graph and is represented by a single number.
The original McCabe metric is defined as follows:

$$\mathbf{CC = e - n + 2}, \text{ where,}$$

CC = the cyclomatic complexity of the flow graph G of the program in which we are interested, e is the number of edges in G and n is the number of nodes in G.
CC can also be calculated as (**no. of decisions + 1)**
There are four basic rules that can be used to calculate CC:

- Count the number of if/then statements in the program.
- Find any select (or switch) statements, and count the number of cases within them. Find the total of the cases in all the select statements combined. Do not count the default or "else" case.
- Count all the loops in the program.
- Count all the try/catch statements.

Adding the numbers from the previous 4 steps together and then adding 1 constitutes thecyclomatic complexity of the given program.

CCM(CyclomaticComplexity Metric) tool
CCM is a tool by Jonas Blunck[10]that analyses java-script, TypeScript, C, C++ and C# code (*.js, *.cs, *.ts, *.c, *.cpp, *.h, *.hpp). It works on both managed and unmanaged code, reportingthe Cyclomatic Complexity. Themetric gives number of independent linear paths through a unit of code and is useful to determine the complexityof *unit* of code (for this particular tool, a unit is a function or a method). Command line version can be output into xml, which allows easy integration with the build software and the continuous Integration software of Visual Studio 2008/2010/2012.
Disadvantages are that it is not a stand-alone tool and it does not provide graphical representation for code analysis.

III. THE PROPOSED APPROACH

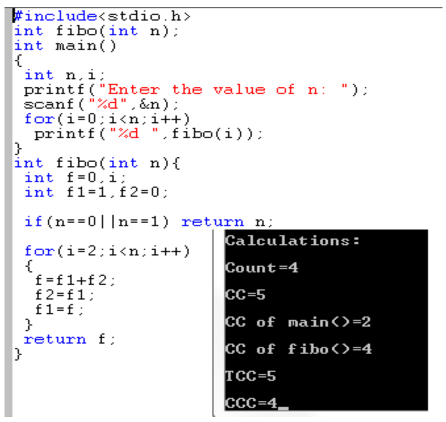
This paper presents an analysis of one of the most used metric that calculate complexity called as cyclomaticcomplexity metric. This paper implements complexity analysis using the two new metrics: Total cyclomatic complexity (TCC), Coupled Cyclomatic Complexity (CCC).
$TCC=CC+\sum TCC(fc_i)-n$
$CCC=CC+\sum\alpha(CCC(fc_i)–n)$, where,
CC: Cyclomatic complexity
α: Coupling value- to represent the intra-modular complexity.

Total Cyclomatic Complexity
TCC gives the value for inter-modular complexity which indicates the measure of interaction between the modules. Total Cyclomatic Complexity (TCC) is derived from McCabe in order to add the summation of all Cyclomatic Complexities computed on all functions that can possibly be executed by a module.

Coupled Cyclomatic Complexity
The interaction between modules has a real impact in measuring complexity[1]. For this reason a new variable is introduced called α coupling value to represent the intra-modular complexity. The value of α coupling value is related to the level of coupling. It takes into account the inter-modular complexity. The Coupled Cyclomatic Complexity (CCC) computes the CC for all the functions called. At each calculation of the CC it is multiplied by the (α coupled value). If α=1 (Data coupling constant), a module passes data through scalar or array parameters to the other module. If α=3 (control coupling constant), a module passes a flag value that is used to control the internal logic of the other.
The constructed Lexical analyzer is executed based on the following steps:
Step 1: Lexical analyzer parses the input code into tokens.
Step 2: The tokens are categorized into symbols, alphabets and digits.
Step 3: The symbols are checked for logical AND or logical OR. If yes, go to step 6 else go to step 4.
Step 4: The tokens are checked for identifiers. If yes, go to step 5 else go to step 7.
Step 5: The tokens are checked for conditional statements such as if, for, while, case, and default. If yes, go to step 6 else go to step 8.
Step 6: The counter variable 'C' is incremented by 1.
Step 7: The tokens are checked for digits.
Step 8: The lexical analyser calculates TCC and CCC based on variable C.

IV. IMPLEMENTATION

The proposed approachis implemented and analysed for three C programs representative of standard synthetic benchmarks such as Fibonacci program, program to find whether the given character is alphabet or digit and Factorial program. The output of the lexical analyzer consists of Count (Number of decisions),Cyclomatic Complexity (CC) of the entire C code, Cyclomatic Complexity (CC) of each module (Main function or any function associated with it), TCC (Total CyclomaticComplexity: Intra-modular Complexity) and CCC (Coupled Cyclomatic Complexity: Inter-modular Complexity)[1]. The code and output of each of these three programs are shown in figure 1.
1) Fibonacci program
In the above test case, there are two modules included and they are main( ) and fibo( ) functions.
CC is calculated for each module based on the formula:



```
#include<stdio.h>
int fibo(int n);
int main()
{
 int n,i;
 printf("Enter the value of n: ");
 scanf("%d",&n);
 for(i=0;i<n;i++)
  printf("%d ",fibo(i));
}
int fibo(int n){
 int f=0,i;
 int f1=1,f2=0;

 if(n==0||n==1) return n;

 for(i=2;i<n;i++)
 {
  f=f1+f2;
  f2=f1;
  f1=f;
 }
 return f;
}
```

```
Calculations:

Count=4

CC=5

CC of main()=2

CC of fibo()=4

TCC=5

CCC=4_
```

Figure 1. Fibbonacci program and the output

CC=no of decision statements+1

The CC calculated for each module is as follows:

CC for main( )= 2

CC for fibo( )=4

TCC= CC+∑TCC(fci)-1

TCC= CC(main)+ ∑TCC(fci)-1

TCC= CC(main)+CC(fibo)-1

TCC=2+4-1 = 5

CCC= CC+∑α(CC(fci)-1)

CCC= CC(main)+∑α(CC(fibo)-1)]-1

CCC= CC(main)+1*[α(CC(fibo)-1)]-1

CCC= CC(main)+1*[1*(4-1)]-1

CCC=2+2

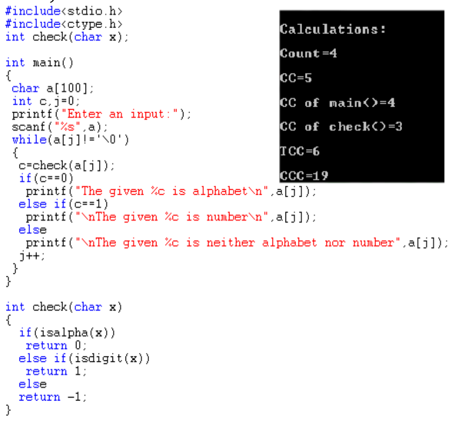
CCC=4

α = Coupling constant between main() and fibo()

α= 1(data)

2) Program to check alphabet or digit in a given string:

In the considered test case shown in figure 2, there are two modules included and they are main( ) and check( ) functions.



```
#include<stdio.h>
#include<ctype.h>
int check(char x);

int main()
{
 char a[100];
 int c,j=0;
 printf("Enter an input:");
 scanf("%s",a);
 while(a[j]!='\0')
 {
  c=check(a[j]);
  if(c==0)
   printf("The given %c is alphabet\n",a[j]);
  else if(c==1)
   printf("\nThe given %c is number\n",a[j]);
  else
   printf("\nThe given %c is neither alphabet nor number",a[j]);
  j++;
 }
}

int check(char x)
{
  if(isalpha(x))
   return 0;
  else if(isdigit(x))
   return 1;
  else
  return -1;
}
```

```
Calculations:

Count=4

CC=5

CC of main()=4

CC of check()=3

TCC=6

CCC=19
```

Figure 2. Checking TCC and CCC

CC is calculated for each module as below.

CC=no of decision statements+1

The CC calculated for each module is as follows:

CC for main( )= 4

CC for check( )=3

TCC= CC+∑TCC(fci)-1

TCC= CC(main)+CC(check)-1

TCC=4+3-1 = 6

CCC= CC+∑α(CC(fci)-1)

CCC= CC(main)+α(CC(check)-1)]-1

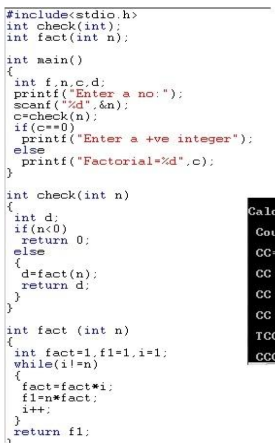
CCC= CC(main)+3*[α(CC(check)-1)]-1

CCC= CC(main)+3*[3*(CC(check)-1)]-1

CCC= 4+3*[3*(3-1)-1]

CCC=19
α= Coupling constant between main() and check()
α= 3(control)

3) Factorial program:



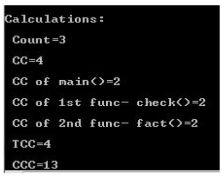
```
#include<stdio.h>
int check(int);
int fact(int n);

int main()
{
 int f,n,c,d;
 printf("Enter a no:");
 scanf("%d",&n);
 c=check(n);
 if(c==0)
  printf("Enter a +ve integer");
 else
  printf("Factorial=%d",c);
}

int check(int n)
{
 int d;
 if(n<0)
  return 0;
 else
 {
  d=fact(n);
  return d;
 }
}

int fact (int n)
{
 int fact=1,f1=1,i=1;
 while(i!=n)
 {
  fact=fact*i;
  f1=n*fact;
  i++;
 }
 return f1;
```

Output



```
Calculations:
 Count=3
 CC=4
 CC of main()=2
 CC of 1st func- check()=2
 CC of 2nd func- fact()=2
 TCC=4
 CCC=13
```

In the above test case, there are three modules included and they are main( ), check( ) and fact( ) functions.
CC is calculated for each module based on the formula:
CC=no of decision statements+1
The CC calculated for each module is as follows:
CC for main( )= 2(-if-)
CC for check( )=2(-if-)
CC for fact( )=2(-while-)
TCC= CC+∑TCC(fci)-1
TCC=CC(main)+[CC(check)+∑TCC(fact)-1]-1
TCC=CC(main)+[CC(check)+CC(fact)-1]-1
TCC=2+[2+2-1]-1 = 4
CCC= CC+∑α(CC(fci)-1)
CCC=CC(main)+αa[CC(check)+∑αb(CC(fact)-1)]-1
CCC=CC(main)+αa[CC(check)+αb(CC(fact)-1)-1]-1
CCC= 2+3[2+3(2-1)-1]-1 = 13
αa = Coupling constant between check() and main()
αa = 3(control)
αb = Coupling constant between main() and fact()
αb = 3(control)

V. CONCLUSION

Analysis of complexity of a software program can be helpful in various development phases resulting in greater business advantages and quality. Cyclomatic approach is being improved when compared to McCabe where the intra-modular metric is being added to the existing inter-modular metric which is an innovative idea presented in this paper.Verification and validation of the proposed approach is carried out. Measuring the complexity in design phase can achieve many advantages in quality, due to the contribution of this phase in reducing the cost and effort of redesign and maintainability. The analysis for interaction between modules that is implemented in our work shows that it would be more efficient.

REFERENCES

[1] AymanMadi, OussamaKassemZein and SeifedineKadry, " On the Improvement of Cyclomatic Complexity Metric", International Journal of Software Engineering and Its Applications Vol. 7, No. 2, March, 2013
[2] Rajiv D. Banker, Srikant M. Datar, Dani Zweig, **"**Software Complexity and Maintainability**"**
[3] T. McCabe, "A Complexity Measure", IEEE Transactions on Software Engineering, December, 1976
[4] T. McCabe, "Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric," NIST Special Publication,pp. 500-235, September, 1996
[5] H. Zuse, "Software Complexity: Measures and Methods", De Gruyter, Berlin, 1991.
[6] S. Nystedt, "Software Complexity and Project Performance", University of Gothenburg, 1999.
[7] K. Geoffrey, "Cyclomatic Complexity Density and Software Maintenance Productivity", IEEE Transactions on Software Engineering, vol. 17, no. 12, December, 1991.
[8] T. Shimeall, "Cyclomatic Complexity as a Utility for Predicting Software Faults",March, 1990.
[9] I. Chowdhury, "Using Complexity, Coupling, And Cohesion Metrics as EarlyIndicators of Vulnerabilities", Queen's University Canada, September, 2009.
[10] Jonas Blunk, "Cyclomatic Complexity Analyser", October 2012.