2012 International Conference on Networks and Information (ICNI 2012)

IPCSIT vol. 57 (2012) © (2012) IACSIT Press, Singapore
DOI: 10.7763/IPCSIT.2012.V57.20

Sequential Cyclomatic Complexity Over a Chain of Function Calls

Kiran Kumar B.[1], Jayadev Gyani[2] and Narsimha G.[3+]

[1]Associate Professor, Dept. of IT, Kakatiya Institute of Technology & Science, Warangal, INDIA

[2] Professor, Dept. of CSE, Jayamukhi Institute of Technological Sciences, Narsampet, INDIA

Member, ACM.
[3]Assistant Professor, Dept. of CSE, JNTUH College of Engineering, Jagityal, INDIA

Abstract. McCabe's Cyclomatic Complexity (CC) is well known software metric which indicates the complexity of a software unit such as a function or an algorithm. It directly measures the number of linearly independent paths through a program's source code. In the existing method of evaluating CC, chain of function calls will not be considered. However, it may not give the true complexity when a function includes other function calls. In this paper, we have modified conventional CC for a chain of function calls. We consider that this will reflect the actual complexity of the function. Further, we have extended this concept to evaluate the complexity of an entire program. We experimentally compared our approach with the conventional CC.

Keywords: Cyclomatic Complexity, software metric, chain of function calls

1. Introduction

Complexity of a unit is used to estimate the effort or cost of a program. It is also used in risk prediction. Complexity is a key factor in project scheduling as an under estimated module may consume more time. Cyclomatic Complexity (also known as conditional complexity) is a software metric used to measure complexity of a software unit. Cyclomatic Complexity is measured as the number of conditional paths available in a software unit [3]. The program with no decision points such as IF, WHILE, or FOR has Cyclomatic Complexity of 1. A program with a single IF statement has a complexity of 2; one for each possible branch. Thomas McCabe published a paper arguing that code complexity is defined by its control flow. Since that time, others have identified different ways of measuring complexity (e.g. data complexity, module complexity, algorithmic complexity, call-to, call-by, etc.). Although these other methods are effective in the right context, it seems to be generally accepted that control flow is one of the most useful measurements of complexity.

To find Cyclomatic Complexity of programs based on McCabe's theory, different commercial tools are available. Metrics plug-in, checkstyle plug-in for eclipse, CCM for C++ and C#, source monitor, Dev metrics for C#, NDepend for .Net platform. All these tools are not considering the chain of function calls. So, we extended CC to consider the chain of function calls.

2. Previous work

Different software metrics were proposed to measure the size and structure of a software system. Cyclomatic Complexity (CC) proposed by McCabe is intended to measure software complexity by examining the software program's flow graph. In practice, CC amounts to a count of the "decision points" present in the software unit. CC can be calculated as:

+ *E-mail address*: kiran_b_kumar@yahoo.com; jayadevgyani@yahoo.com; narsimha06@gmail.com

CC = E – N + 2P

where E is the number of edges, N is the number of nodes, and P is the number of discrete connected components.

Lines of Code (LOC) is any line of program text that is not a comment or a blank line, regardless of the number of statements or fragments of statements on the line [1]. This specifically includes all lines containing program headers, declarations, and execuTab. and non-execuTab. statements. It is easy to understand, fast to count, and independent of the programming language. Halstead Complexity Metric is based on the volume [2]. According to Halstead: a computer program is an implementation of an algorithm considered to be a collection of tokens which can be classified as either operators or operands. Let n1 be the number of distinct operators, n2 be the number of distinct operands, N1 be the total number of occurrences of operators and N2 be the total number of occurrences of operands of a program (P).

The volume of a program P is defined as

V = N*log (n) where N = n1log (n1) +n2log (n2).

Different commercial tools were developed based on McCabe's theory. Metrics plug-in handles, calculating McCabe's Cyclomatic Complexity, efficient coupling, lack of cohesion methods, lines of code in method, number of fields, number of locals in scope, number of parameters, number of statements, weighted methods for class [4]. CCM is a tool that analyzes C, C++, C# and javascript code and produces a list of the most complex methods or functions in the analyzed code base [5]. NDepend is a static analysis tool for .NET managed code [6]. This tool supports a large number of code metrics, allows for visualization of dependencies using directed graphs and dependency matrix.

The tools also performs code base snapshots comparison, and validation of architectural and quality rules. User-defined rules can be written using LINQ queries. This possibility is named CQLinq. The tool also comes with a large number of predefined CQLinq code rules. Code rules can be checked automatically in Visual Studio or during continuous integration. SourceMonitor collects metrics in a fast, single pass through source files, measures metrics for source code written in C++, C, C#, VB.NET, Java, Delphi, Visual Basic (VB6) or HTML [7]. It includes method and function level metrics for C++, C, C#, VB.NET, Java, and Delphi. It offers Modified Complexity metric option, saves metrics in checkpoints for comparison during software development projects, displays and prints metrics in Tab.s and charts, including Kiviat diagrams. It operates within a standard Windows GUI or inside your scripts using XML command files and exports metrics to XML or CSV (comma-separated-value) files for further processing with other tools. CheckStyle plug-in for eclipse provides checking of code layout issues [8], and also provides checks that find class design problems, duplicate code, or bug patterns like double checked locking. devMetrics is a C# code analysis tool that gathers software metrics [9], so that developers, leads and software managers can quickly identify potentially problematic and high-risk areas of their .NET code. devMetrics analyzes C# code measuring size and complexity. All these tools are not considering the chain of function calls in the calculation of Cyclomatic Complexity.

3. Proposed Work

We propose Sequential Cyclomatic Complexity (SCC), which considers chain of function calls while calculating the Cyclomatic complexity. SCC assumes CC of the function and CC of functions which are called from it. So,

$$SCC(\text{fn}) = CC(fn_{cg}) + \sum_{i=1}^{n} CC(fn(i)) \quad (1)$$

where fn_{cg} is calling function, fn_{cd} is called function, n is number of function calls.

Sequential Cyclomatic Complexity of the program is equal to sum of Sequential Cyclomatic Complexity of all the functions of the program.

$$SCC(program) = \sum_{j=1}^{m} SCC(fn(i)) \quad (2)$$

where m is the number of functions.

For example, consider the following C program fragment.

```
void  fun1()
{
  if(condition)
    fun2();
  else
    fun3();
}

void fun2()
{
  printf("In fun2");
}
void fun3()
{
  printf("In fun3");
}
void main()
{
  fun1();
}
```

As per the McCabe's approach, the Cyclomatic Complexity of a unit is number of predicate nodes +1. So, the Cyclomatic Complexity of main is 1, fun1 is 2, fun2 is 1, fun3 is 1. But the function fun1 is calling fun2 and fun3, so, as per Eq.1, Sequential Cyclomatic Complexity of fun3 is 1, fun2 is 1, fun1 is 4, and main is 5. As per the Eq.2, the Sequential Cyclomatic Complexity of the program is 11. In case of recursive function calls, we have considered that function complexity one additional time, because number of times a function called itself during recursion is not known at compile time.

4. Experimental Results

We developed a tool using C#.Net for finding Sequential Cyclomatic Complexity of programs written in C language. We experimented the execution of the tool on three C programs, i) Evaluation of arithmetic expression, ii) Chain of matrix multiplication, and iii) Traveling salesman problem. The results show that the calculation of Cyclomatic Complexity by considering chain of function calls gives true complexity measure of the program. The comparison between Cyclomatic Complexity and Sequential Cyclomatic Complexity for Traveling Salesman problem is shown in tabular and pictorial representation.

The following Tab. shows the number of predicate nodes of each function.

Tab. 1:Functions and the count of their predicate nodes

Function Name	Predicate nodes	No. of Occurrences
Mincost	If	1
Least	For	1
Least	If	4
Get	For	4
Put	--	--
Main	--	--

From Tab. 1, we calculated Cyclomatic Complexity of each function using McCabe's approach and program complexity by adding individual Cyclomatic Complexity of functions. Tab. 2 shows the functions and their Cyclomatic Complexity and overall program Cyclomatic Complexity.

Tab. 2:Functions with their Cyclomatic Complexity

Function	CC
Mincost	2
Least	6

Get	5
Put	1
Main	1
Program	15

Then we traced the program to find out the calling and called functions. Tab. 3 shows the details.

Tab. 3:Called and Calling functions

Calling Function	Called Function	No. of times called
Mincost	Least	1
Mincost	Mincost	1
Least	--	--
Get	--	--
Put	--	--
Main	Mincost	1
Main	Put	1
Main	Get	1

Then we calculated the Sequential Cyclomatic Complexity of each function using Eq.1 and program complexity using Eq. 2.

From Tab. 3, we can find that

SCC (least) = CC (least). So, SCC (least) = 6.
SCC(mincost)=CC(mincost)+CC(least)+CC(mincost). So, SCC(mincost)=2+6+2=10
[As mincost() is a recursive function so, we considered it's CC twice]
SCC(main)=CC(main)+CC(mincost)+CC(put)+CC(get). So, SCC(main)=1+2+1+5 = 9

Tab. 4 shows functions and their Sequential Cyclomatic Complexity and cumulative Sequential Cyclomatic Complexity of the program.

Tab. 4. SCC and cumulative SCC

Function Number	Function	Sequential Cyclomatic Complexity(SCC)
1	mincost	10
2	least	6
3	get	5
4	put	1
5	main	9
Cumulative Cyclomatic Complexity of the program		31

Fig.1 shows the comparison between Cyclomatic Complexity and Sequential Cyclomatic Complexity of the traveling sales man program. Fig.2 depicts the comparison of cumulative Cyclomatic Complexity and cumulative Sequential Cyclomatic Complexity of the program.

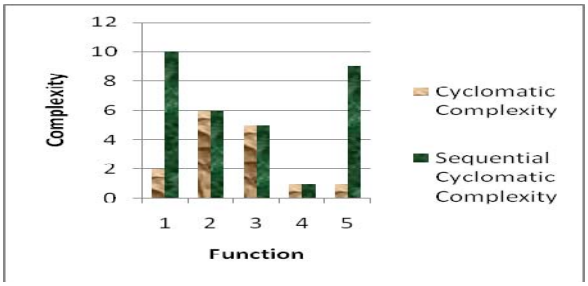

Fig.1:Comparison between CC and SCC of the functions of TSP

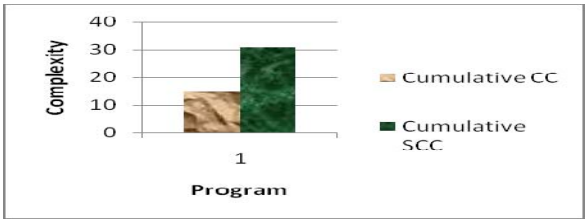

Fig. 2:Comparison of Cumulative CC and Cumulative SCC of TSP

5. Conclusions and future work

Our experimental results reflect the true Cyclomatic Complexity of the program where we have considered the chain of function calls. In this version, we assumed that whenever there are recursive function calls we add that function complexity one additional time. It simplifies the process of evaluating SCC. Moreover, number of times a function calling itself during recursion is not known at compile time. In future work, we will extend this work to object-oriented programming languages.

6. References

[1] Conte S.D., Dunsmore H.E. and Shen V.Y., "Software Engineering Metrics and Models". Benjamin- Cummings Publishing Co., Inc. Redwood City, CA, USA, 1986.

[2] Halstead, Maurice H. "Elements of Software Science". Amsterdam: Elsevier North-Holland, Inc. 1977.

[3] McCabe T.J., "A complexity metric". IEEE Transactions on Software Engineering, 2(4): 308-320, 1976.

[4] www.metrics.sourceforge.net

[5] www.blunck.info/ccm.html

[6] www.ndepend.com

[7] www.compwoodsw.com/soursemonitor.html

[8] www.checkstyle.sourseforge.net

[9] www.devmetric.com

Kiran Kumar Bejjanki has completed his Master of Computer Applications from Kakatiya University in 1998, M.Tech in Computer Science & Engineering in 2010, and pursuing his Ph.D in Computer Science & Engineering in JNTU, Hyderabad in INDIA. He is a life member of ISTE. His areas of research are Software Engineering and Data Mining.

Dr.Jayadev Gyani has completed his B.Tech. in Mechanical Engineering from Kakatiya University in 1991, M.Tech. in Computer Science & Engineering in 1994 from Osmania University and Ph.D. in Computer Science in 2009 from University of Hyderabad, INDIA. He is a member of ACM, ISTE and Senior Member of IACSIT. His areas of research are Software Engineering, Cloud Computing, Wireless Sensor Networks and Video Content Analysis.

Dr.Narsimha Gugulothu has completed his B.E. in Electronics and Communication Engineering from Osmania University in 1996, M.Tech in Computer Science and Engineering from Osmania University in 1999 and Ph.D. in Computer Science and Engineering in 2009 from Osmania University in INDIA. He is a life member of ISTE. His areas of research are Networks and Data Mining.