Cyclomatic Complexity for WCF: A Service Oriented Architecture

Mir Muhammd Suleman Sarwar, Ibrar Ahmad, Sara Shahzad
Department of Computer Science
University of Peshawar
Peshawar, Pakistan
mirsuleman1@gmail.com, ibrar@upesh.edu.pk, sara@upesh.edu.pk

***Abstract*—SOA is a latest architecture for softwares and a lot of tools are available to implement this architecture.**
Critics of cyclomatic complexity argue that complexity changes with modularization of code. If the technology is shifted from linear programming to OOP and SOA, the complexity of code will also change. The cyclomatic complexity was for the first time introduced by TJ McCabe as metric for the measurement of complexity of a piece of code. McCabe calculated the complexity of a sample code written in fortran language. Fortran is a linear programming language and there are no functions and classes in this language. Hence at the time of introducing cyclomatic complexity there was no concept of structured languages and object oriented languages. This is ignored in McCabes's cyclomatic complexity due to which it is not enough to measure complexity for advance programming architectures like OOP and SOA. Further work has been done on the complexity of structured, OOP and SOA but still work is required on SOA.This study proposes a new metric for the measurement of complexity of WCF a SOA. The significance of this new metric is that it can help to estimate cost of a new project, maintenance cost of already existing projects, basis path testing, comparison of two projects and many other factors.

Keywords: Complexity, Service-Oriented Architecture, Object Oriented Paradigm, Windows Communication Foundation

I. INTRODUCTION

A critical question facing software engineering is how to modularize a software system that is both testable and maintainable. Thomas J. McCabe, Sr. formularized a mathematical technique that provides the quantitative basis for modularization to identify software modules that are difficult to test or maintain [1]. This technique is based on control flow. Before this technique physical size that is lines of code (LOC) were considered which is not adequate because a 50 line of code consisting of 25 consecutive "If than" constructs may have 33.5 million distinct control paths, only a small percentage of which would probably ever be tested. McCabe's technique only considered decision control statements such as "If else" and "loops" etc.

Cyclomatic complexity is used to calculate the complexity of a program's source code. It is also called conditional complexity. This technique measures all the linearly independent paths of a program's source code by creating a control flow graph. In this control flow graph the nodes are actually indivisible groups of commands which are connected by edges. These edges connect nodes in the same sequence in which the commands are in original program's source code. Complexity is calculated using this control flow graph [1].

McCabe also used cyclomatic complexity for testing of code. He called it basis path testing. In this technique each linearly independent path was tested and the number of test cases were equal to cyclomatic complexity [1].

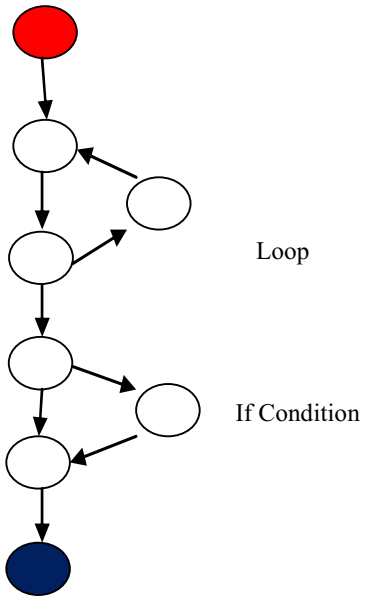

Fig. 1. *Decission Control Graph*

Every decision control statement is considered a node attached with a number of nodes with possible distinct paths . For example loops are presented by three nodes connected in a cycle and if statements are represented by three nodes in two different directions. The nodes were than combined with edges and complexity (cyclomatic complexity) is calculated by the following formula.

978-0-7695-4927-9/12 $26.00 © 2012 IEEE
DOI 10.1109/FIT.2012.39

$$V(G) = e - n + 2 \quad (1)$$

Since there is only one predicate (strongly connected component) therefore, no "P" is written with constant "2" in the formula because value of "p" is 1.

McCabe also calculated the cyclomatic complexity for more than one predicates.

$$V(G) = e - n + 2P \quad (2)$$

Where "p" represents predicates or strongly connected components such as those in structured programming. Also he gave an equation to calculate complexity with the help of predicates.

$$V(G) = \pi + 1 \quad (3)$$

Where π represents predicates [1].

II. TWO DIMENSIONS OF RESEARCH WORK ON CYCLOMATIC COMPLEXITY

The research done on cyclomatic complexity can be broadly categorized in two dimensions, the uses of cyclomatic complexity and the improvements in cyclomatic complexity. The former refer to the applications of cyclomatic complexity and later means advancement in cyclomatic complexity such that it can be applicable on a new architecture. Both of these have a direct relationship in such a way that improvements may increase the uses of cyclomatic complexity.

A. Uses

Following section describes different applications of cyclomatic complexity.

Zhang Zhonglin and Mei Lingxia used basis path testing as an important method of white box testing. They used cyclomatic complexity to define a basic set of feasible paths, which is then tested by white box testing [2].

Results of a study based on statistical techniques shows that a higher complexity is associated with a lower design quality of code and hence higher maintenance costs [3].

In the year 1996 a new problem was identified i.e "The Year 2000 problem". In most of the softwares at that time two digits were reserved for year field. So 2000 was written as 00, which overlaps with 1900. So there was a problem in stating the date which in turn raised many other problems for companies. The cost of maintenance for fixing this problem was calculated but most of the companies don't know the quality of there dates portfolio or how to test the date transformation. Cyclomatic complexity deals with the inherent complexity of the date change itself. Cyclomatic complexity quantifies the risk of date changes and derive the exact tests needed to verify the changes in date [4].

Another research presents evidence on the basis of statistical techniques that LOC and cyclomatic complexity have stable practically perfect linear relationship that holds across programmers, languages, code paradigms (procedural versus object oriented) and software processes [5].

Sanjay Misra and Ferid Cater used cyclomatic complexity as a metric to measure the complexity of code in Python language [6].

B. Improvements

Following section describes improvements to the cyclomatic complexity metric.

Studies have shown that existing metrics consistently fail to capture complexity or cohesion. C. Ikerionwu in his work shows that there is an overlap between some of the complexity and cohesion metrics and pointed to a more basic relationship between complexity and cohesion: that a lack of cohesion may be associated with high complexity. In his study the cohesion of a set of statements S is defined as [7]:

$$C(S) = \frac{|S| \dim(s)}{|G| \dim(G)} \quad (4)$$

Cohesion of a module is given as:

$$C(M) = \frac{\sum C(Ri)}{n} \quad (5)$$

A study criticized cyclomatic complexity that lines of code other than decision control statements were ignored. Also, the number of paths through any software with a backward branch is potentially infinite. Myer's criticized cyclomatic complexity because of its failure to distinguish between selections with and without "Else" branches [8].

Another reason of criticism on cyclomatic complexity is the treatment of case statements. Hansen has suggested that since they were easier to understand than the equivalent nested IF's they should only contribute one to the module complexity [9].

Other researchers have suggested $\log_2(n)$ relationship where n is the number of cases [10].

Nasib S. Gill and Sunil Sikka proposes a complexity model for classes in object oriented systems [11].

The model computes Class Complexity (CC) as a sum of Method Complexity (MC). MC is further computed as a sum of Control Flow Complexity (CFC), Total Method Call Complexity (TMCC) and Total Data Call Complexity (TDCC).

Class Complexity(CC)=∑MC for each method in the class.

Method Complexity (MC) = Control Flow Complexity (CFC) +Total Method Call Complexity (TMCC) + Total Data Call Complexity (TDCC)

Control Flow Complexity(CFC) = McCabe's Cyclomatic Complexity.

Total Method Call Complexity is given as:

$$TMCC = \begin{cases} 0 & if\ no\ method\ call \\ \sum MCC & otherwise \\ for\ each\ method\ call \end{cases} \quad (6)$$

Total Data Call Complexity is given as:

$$TDCC = \begin{cases} 0 & if\ no\ data\ call \\ \sum DCC & otherwise \\ for\ each\ data\ call \end{cases} \quad (7)$$

The following figure describes diagrammatically the above equations.

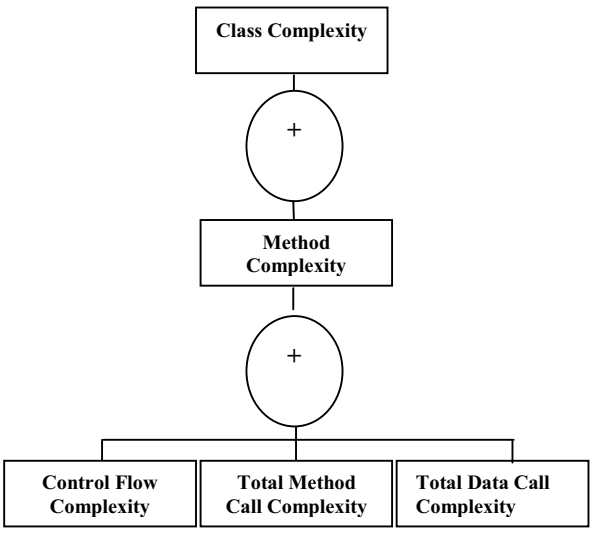

Fig. 2. *Class Complexity*

III. WCF A SERVICE ORIENTED ARCHITECTURE(SOA)

"The application designed by SOA is based on services that interact with business components. Each service defines a particular business function. These services interact with each other to accomplish business processes" [12] .There are multiple tools available to implement SOA however all of these have the following two main parts.

1. The service program.
2. The client-side program

As an example here we take Windows Communication Foundation (WCF) of Dot.Net Framework 4.0. Figure 3 describes parts of WCF, which are explained as follows.

A. The Client-side Application

The client-side program consists of a source code and an attached xml file which has all the necessary references for the service. The reference of this xml file is than given in the source code [13].

B. The Service

Service program has the following parts [13].

1. .svc file.
2. .xml file.
3. Service contract.
4. Data contract.

The .svc file defines a service and serves as an end point to a client program/user.

The xml file contains the ABC(Address, Binding and Contract) of a WCF SOA program. The ABC is a way to understand easily how WCF works.

Address is where to communicate. This is the URL that will be used internally to map requests and responses.

Binding is how to communicate. This is the protocol that the server and clients agree for communication e.g. TCP, BasicHttp, MSMQ, Named pipes and many others.

A contract is an agreement of terms i.e what will be communicated. There are two main types of Contracts.

1. Service Contract.
2. Data Contract.

A Service Contract is the API of service which client program calls. The Service Contract consists of two parts, a Service Contract and a Service Behavior. The Service Behavior is a class and the Service Contract is an interface.

A Data Contract is the data that will flow between client application and service. The Data Contract has two classes. One for entertaining or receiving the request and the other for giving response.

IV. COMPLEXITY FOR WCF

Our scope of study is to calculate the complexity of WCF and not concerned with pros and cones of this architecture over others or how it is more effective and efficient than linear programming, structured programming and OOP. However, the calculation of complexity may reflect these factors in such a way that a higher complexity shows increased cost and a less complexity shows a decreased cost. Which can be used to compare different architectures. In order to calculate the complexity, the difference in the nature of the code is considered. So, here the advantages and disadvantages of SOA is not considered and will not be discussed. Instead nature of code is considered such that if the code is different from the

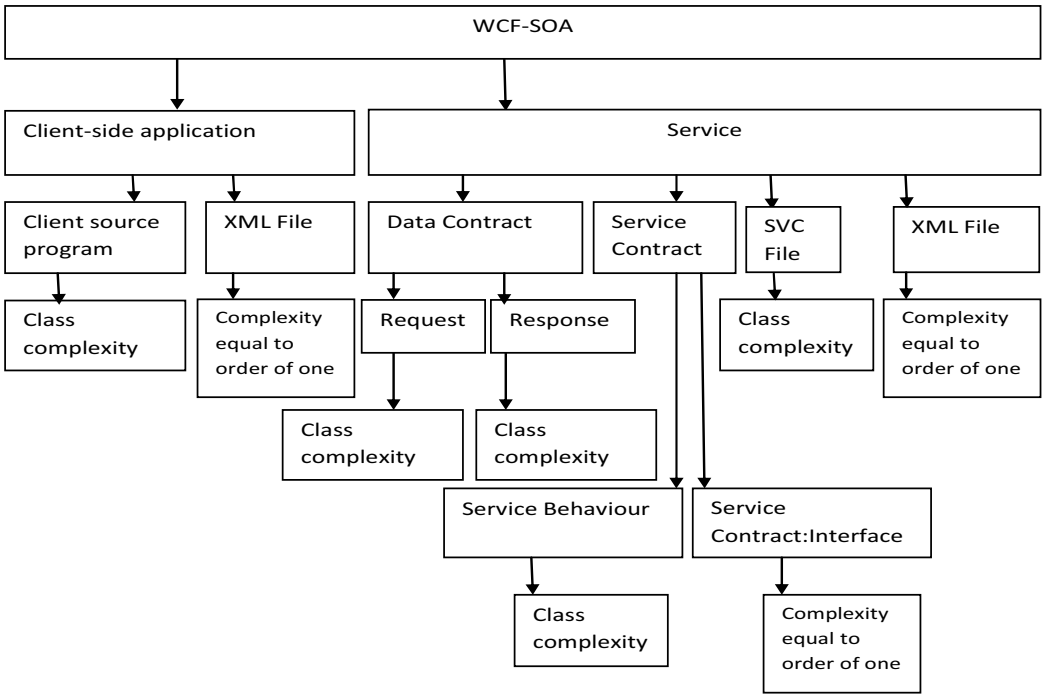

Fig. 3. *WCF Complexity*

previous architectures than the way of calculating complexity for SOA must be different from the rest, but if there is no difference than there is no need for the formulation of a new equation to calculate complexity of SOA. Figure 3 describes all the components of WCF and their complexity.

A. Complexity for Client-side Program

The xml file does not consist of any decision control statement and is a separate file than the original source code. So, in a client-side application the complexity of the xml file will be equal to order of one. The rest of the code is object oriented and its complexity will be same as that of class complexity proposed by Nasib S. Gill and Sunil Sikka [11] as in Fig. 2.

B. Complexity for the Service Program

The complexity of the .svc file will be equal to class complexity as it is a class.

The complexity of the xml file will be equal to order of one as discussed above.

The complexity of the Data contract request and receive classes will also be equal to class complexity since both are classes. Also the complexity of Service Behavior class of Service Contract will be equal to class complexity since it's a class. But the complexity of the Service Contract cannot be calculated by the class complexity since its not a class. Also its complexity cannot be calculated by cyclomatic complexity since it does not have any decision control statements or strongly connected components. Also it is not very complex and has very few lines. The complexity of the Service Contract Interface will be equal to order of one.

The combined complexity of all the components of WCF-SOA will be the sum of class complexity of client-side program and service program. So, the overall complexity will be equal to the sum of complexities of client source program and xml file, the request class and response class of Data Contract of service program, the Service behavior and Service Contract Interface of the Service Contract, the .svc and xml file of service program.

Total Complexity of program= Complexity of(Client source program+Data Contract. Request Class + Data Contract. Response Class + Service Contract. Service Behavior Class + .svc file complexity).

On the basis of analysis of WCF structure, the following algorithm is proposed to calculate the complexity. The pseudocode of the proposed algorithm is as follows:

Algorithm : COMPLEXITY

N ← Node
E←Edge
P←Predicate
M←Cyclomatic Complexity
TMCC←Total Method Call Complexity
TDCC←Total Data Call Complexity
CFC←Control Flow Complexity
MC←Method Complexity
CC←Class Complexity

DCRequestC←Data Contract Request Class Complexity
DCResponseC←Data Contract Response Class Complexity
SBC←Service Behaviour Class Complexity
CSC←Client-side Class Complexity
SC←Service Complexity
SOAC←SOA Complexity
SFC ← SVC File Complexity

CyclomaticComplexity(Method code)

1. N ← 0
2. E←0
3. P ← 0
4. for i ← 1 to Number Of P
5. if DCS← **If or Switch**
6. N ← N+3
7. E←E+3
8. Connect three nodes in two different directions
9. if DCS← **Loop**
10. N ← N+3
11. E←E+3
12. Connect three nodes in a cycle
13. P←P+1
14. M←E- N + 2P
15. return M

ClassComplexity(Class code)

1. TMCC←0
2. TDCC←0
3. CFC←0
4. MC←0
5. CC←0
6. for j ←1 to Number Of Methods
7. for k←1 Number Of Method Call
8. TMCC←TMCC+1
9. for l← Number of Data Call
10. TDCC←TDCC+1
11. CFC ← CyclomaticComplexity(Method code)
12. MC ← MC+CFC+TMCC+TDCC
13. CC ← MC+MC
14. return CC

SOA Complexity(WCF code)

1. DCRequest←0
2. DCResponse←0
3. SBC←0
4. CSC←0
5. SFC←0
6. Service Complexity←0
7. SOA Complexity←0
8. for i ← 1 to NumOfServices
9. DCRequest ← ClassComplexity (DataContract.Request Class code)
10. DCResponse ← ClassComplexity (DataContract.Response Class code)
11. for j ← 1 to NumOfServiceContract
12. for k ←1 to NumOfServicebehaviors
13. SBC ← ClassComplexity(Service Behavior Class code)
14. SBC←SBC+SBC
15. for l ← 1 to NumOfClients
16. for m ← 1 to NumOfClasses
17. CSC ← ClassComplexity(ClientSideClass code)
18. CSC←CSC+CSC
19. SFC ← Class Complexity(SVC File Class code)
20. SC ← DCRequest+DCResponse+SBC+SFC
21. SOAComplexity ← SC + CSC
22. return SOAComplexity

The cyclomatic complexity function receives input of Method code. The function checks code for all number of predicates(strongly connected components) line by line. For every predicate the function only detects decision control statements(DCS). The first if statement line#5 checks if the DCS is an if statement or switch. If this condition is true three nodes are created and connected to each other in two different directions. The second if statement in line#9 checks if the DCS is a loop. In this case also three nodes and three edges are created but connected in a cycle. After doing this for all predicates the total number of nodes, edges and predicates are put in Eq. 2 to calculate cylomatic complexity. This calculates cyclomatic complexity for control flow complexity of a method.

Next the class complexity function receives input of class code. This function calculates complexity of a class. The complexity of a class is the sum of complexities of all its methods. The complexity of a method is sum of control flow complexity(CFC), total method call complexity(TMCC) and total data call complexity(TDCC). In line#6 the loop runs to total number of methods. Then for every single method another loop executes to total number of method calls in line#7 and 8 which calculates TMCC as in Eq. 6 . Similarly for every method another loop executes to total number of data calls(TDCC) in line#9 and 10 which calculates TDCC as in Eq. 7. Then in line#11 the control flow complexity(CFC) of every method is calculated by calling cyclomatic complexity function with method code as parameters. At line#12, at the end of loop all the values of CFC,TMCC and TDCC are added and put to method complexity(MC) along with previous values of MC in an iterative manner. Finally at line#13 the sum of all values of MC is actually the value of class complexity(CC).

The final function calculates complexity of overall program which receives WCF code as an input. This program may contain multiple services. So, first of all a loop executes for all services in line#8. Then for every service, the complexity for data contract request class(DCRequest) and data contract response class(DCResponse) is calculated by calling the class complexity function by passing codes of request and response classes as parameters in line#9 and 10. The class complexity function returns complexity of DCRequest class and DCResponse class. A service contains Service Contracts which might contain multiple Service Behaviors. So, by calling the class complexity function, the complexity is calculated of all the Service Behaviors(SBC) for all the Service Contracts in line# 11,12,13 and 14. A program may

contain multiple clients and each client program may contain multiple classes. So, complexity for all the client-side classes is calculated in line#15, 16, 17 and 18. At line#19 complexity of SVC file(SFC) is calculated. At line#20 complexity of service is calculated by adding complexities of DCRequest, DC.Response, SFC and SBC. At the end complexities of service program(SC) and client-side program(CSC) are added to get total complexity of WCF-SOA.

V. Conclusion

With the help of this technique we can now calculate the complexity of code for WCF. Also once the complexity has been calculated we can use it to estimate the production cost and maintenance cost, acquire basis path testing for white box testing, quality of design and many other uses of complexity for WCF-SOA.

VI. Limitations and future work

A generalized complexity metric must be formulated which will not only support complexity measurement of all existing architectures but also for any future architectures.

I. References

[1] T. J. McCabe, "A Complexity Measure," IEEE Transactions on Software Engineering, vol. SE-2, pp. 308-320, 1976.

[2] Z. Zhang and L. Mei, "An improved method of acquiring basis path for software testing," 2010 5th International Conference on Computer Science and Education (ICCSE), 2010, pp. 1891-1894.

[3] Francalanci, Chiara and Merlo, Francesco, "The Impact of Complexity on Software Design Quality and Costs: An Exploratory Empirical Analysis of Open Source Applications" (2008). ECIS 2008 Proceedings. Paper 68

[4] T. McCabe, "Cyclomatic complexity and the year 2000," IEEE Software, vol. 13, pp. 115-117, 1996.

[5] G. Jay, J. Hale, R. Smith, D. Hale, N. Kraft and C. Ward, "Cyclomatic Complexity and Lines of Code: Empirical Evidence of a Stable Linear Relationship," Journal of Software Engineering and Applications, Vol. 2 No. 3, 2009, pp. 137-143.

[6] S. Misra and F. Cafer, "A Software Metric for Python Language," Computational Science and Its Applications – ICCSA 2010. vol. 6017, D. Taniar, et al., Eds., ed: Springer Berlin / Heidelberg, 2010, pp. 301-313.

[7] C. Ikerionwu, "Cyclomatic Complexity as a Software Metric, " International Journal of Academic Research; May2010, Vol. 2 Issue 3, p117

[8] G. J. Myers, "An extension to the cyclomatic measure of program complexity," SIGPLAN Not., vol. 12, pp. 61-64, 1977.

[9] W. J. Hansen, "Measurement of program complexity by the pair: (Cyclomatic Number, Operator Count)," SIGPLAN Not., vol. 13, pp. 29-33, 1978.

[10] M. Shepperd, "A critique of cyclomatic complexity as a software metric," Software Engineering Journal, vol. 3, pp. 30-36, 1988.

[11] N. S. Gill and S. Sikka, "New complexity model for classes in object oriented system," SIGSOFT Softw. Eng. Notes, vol. 35, pp. 1-7, 2010.

[12] Brian Travis, MSDN, http://msdn.microsoft.com/en-us/library/aa302164.aspx, Aug 2011.

[13] Sidhartha Gundavarapu, Code Project, http://www.codeproject.com/Articles/18589/Writing-your-first-WCF-Service, June 2011.