A Framework for Improving the Concept of Cyclomatic Complexity in Object-Oriented Programming

Thesis submitted in partial fulfillment of the requirements for the award of degree of

Master of Engineering
in
Software Engineering

Submitted By
Ankita
(801231003)

Under the supervision of:
Vinod Kumar Bhalla
(Assistant Professor)

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT
THAPAR UNIVERSITY
PATIALA – 147004

June 2014

CERTIFICATE

I hereby certify that the work which is being presented in the thesis entitled, "*A Framework for Improving the Concept of Cyclomatic Complexity in Object Oriented Programming*", in partial fulfillment of the requirements for the award of degree of Master of Engineering in *Software Engineering* submitted in Computer Science and Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of *Mr Vinod Kumar Bhalla* and refers other researcher's work which are duly listed in the reference section.

The matter presented in the thesis has not been submitted for award of any other degree of this or any other University.

Ankita

(Ankita Garg)

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

Vinod K Bhalla

(Mr Vinod Kumar Bhalla)

Assistant Professor

Computer Science Department

Countersigned by

(Dr. Deepak Garg)

Head

Computer Science and Engineering Department

Thapar University

Patiala

(Dr. S. K. Mohapatra)

Dean (Academic Affairs)

Thapar University

Patiala

ACKNOWLEDGMENT

First of all, I would like to thank the Almighty, who has always guided me to work on the right path of the life. Due to mercy of God, it has been possible for me to reach so far. This work would not have been possible without the encouragement and valuable guidance of my supervisor Mr. Vinod K. Bhalla, Assistant Professor, Thapar University, Patiala. I thank my supervisor for his time, patience, discussions and valuable comments. I am equally grateful to Dr. Deepak Garg, Associate Professor and Head, Computer Science and Engineering Department, for motivation and inspiration that triggered me for the thesis work. I am also thankful to the entire faculty and staff members of Computer Science and Engineering Department for their direct-indirect help, cooperation, love and affection, which made my stay at Thapar University memorable.

I will be failing in my duty if I don't express my gratitude to Dr. S. K. Mohapatra, Senior Professor and Dean of Academic Affairs the University, for making provisions of infrastructure such as library facilities, computer labs equipped with net facilities, immensely useful for the learners to equip themselves with the latest in the field.

Ankita

(801231003)

ABSTRACT

Zuse defines software complexity as “the software which is difficult to understand and change in future. Mainly three metrics LOC, Halstead’s and cyclomatic complexity are used to measure the complexity of software. There are some problems with LOC and Halstead’s metric. So, to overcome problems, McCabe’s Cyclomatic Complexity was introduced which is the strongest metric among all.

But, there was no concept of object oriented at the time when Cyclomatic complexity was introduced. Because of ignorance of this concept in Cyclomatic complexity it was not considered as a strong metric to measure the complexity of programming language like OOPs. Cyclomatic complexity is not considering or measuring the exact software complexity means if there is interaction between two or three object classes in software then it does not calculates that complexity. There is a need to consider that object coupling complexity in McCabe cyclomatic complexity.

Firstly, there is a need to consider the concept of coupling. That coupling complexity also needed to be in consideration because a change in one affects other. So, in thesis work an algorithm is purposed to measure the interaction between object classes based on all the unique external class references and along with an approach is proposed to improve the concept of cyclomatic complexity.

TABLE OF CONTENT

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 Motivation

Software has an importance in every field such as in law, entertainment, health etc. but good software is that which makes available at low cost and with-in a time [1]. Software development passes through various phases during its development like requirement engineering, planning, design phase which includes low-level design phase or high level design phase and then testing, maintenance phase.

Software development phase includes some more steps also, so that software makes available with-in time, at low cost and with best quality as possible.

Assign the work to the person working in the project to get the productivity. The purpose is to increase the output means productivity by reducing the effort. So, Pareto principal is applied based on 80/20 rule [2]. Some sequences of steps are followed to develop the software and before proceeding to the next step, previous step should be complete. The steps followed are Requirement Analysis, Design, coding, testing and in last maintenance phase. Requirement phase involves regarding gathering all the requirement related to software or project. The requirements can be functional and non-functional type. Design phase consists of high-level design and low-level design and it includes Flow charts, Software architecture, Database architecture, Use-case diagrams, Procedures etc. The most important phase is testing phase, in this step testing is performed to find the errors and then corrective action is taken against these errors. Corrective action taken is not the part of testing. Testing process involves only detecting the errors.

Now process model is taken in to consideration which consists of three sectors like sector A, sector B and sector C.

1. **Sector A:** Engineers and researchers use goals and policies to develop the process and these used goals and policies are risk-free. Software engineers also use various evaluation tools to improve and evaluate the process description.

2. **Sector B:** This step involves managing the software processes. Managers adopt the description developed in step engineering.
3. **Sector C:** In this step, engineers use the tools to develop the application development part and there is a similarity between tools use in this part and tools used in first and second part but the only difference is that tools operates in this part on software parts rather than on process parts [3].

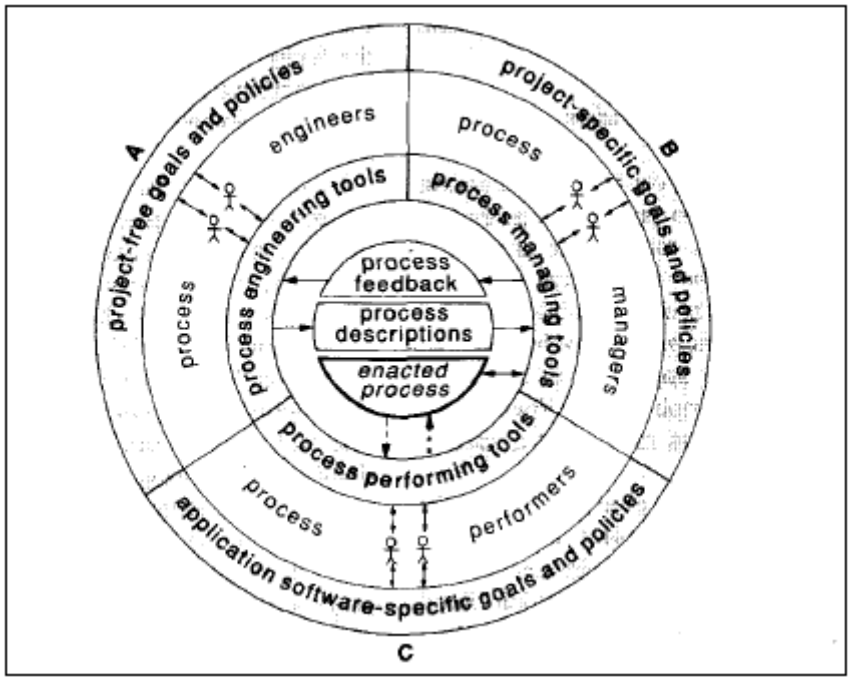


Fig.1.1 The process cycle [3].

Software Engineering includes

1. Requirement engineering
2. Design
3. Quality
4. Architecture of software
5. Software construction
6. Software development
7. Formal methods
8. Software processes.

Object oriented approach is proposed for the development of software system to reduce the cost and to improve the productivity of software system but it lacks and ignores the technique of testing.

So, during software development life cycle two techniques are importantly used to insure the quality and reliability of software.

1. Software testing
2. Software metrics

Software testing is a term which is used to find the errors during software development.

Software metrics is used to improve the quality of software by monitoring the complexity of software. A high quality and error free software have characteristics of understandability, easy to measure and it is considered that risk is high if software is complex. That's why software metric concept has received much attention over the years [3].

1.2 Background

With the advancement of software development the demand for software is increasing day by day and to meet these requirements the complexity of the software goes on increasing. The prime focus of software engineers and researchers is to get quality software. For that purpose it is very essential to measure the software complexity and high complexity software difficult to understand, read and hence, troublesome to change in future. Complex software considered to be the reason for the presence of defects, this leads to consider that software complexity is responsible for poor software quality.

Mainly three software complexity metrics LOC, Halstead's measure of complexity and cyclomatic complexity metrics are used but there are some drawbacks of LOC and Halstead's measure complexity metric and to overcome these problems strongest metric cyclomatic complexity metric was introduced among them and three methods are used to measure the cyclomatic complexity. Mainly cyclomatic complexity is calculated from the control flow graph which consists of edges and nodes and there is a significance of this calculated cyclomatic complexity number.

This number should be in between 1-10 and should not be more than 20. Cyclomatic complexity number greater than 10 signifies that software has high complexity and there will be a chance of more errors in that software. One popular eclipse plug-in named "Metric" is used to calculate the Cyclomatic Complexity. This "Metric" Plug-in provides a number of features like total of classes, total of methods, overridden methods, Depth of inheritance tree, total lines of code, interfaces, packages, specialization index and many other features along with McCabe Cyclomatic Complexity for a particular project.

1.3 Software Complexity

Zuse defines software complexity as "the software which is difficult to understand and change in future" [4]. According to Programmer complexity can be defined as "Difficult to perform task such as testing, maintenance, debugging and to change the software in future" [5] [6]. IEEE defines software complexity as "the degree to which a system or component has a design or implementation that is difficult to understand and verify" [7]. To measure the software complexity various software metrics are used like LOC, Halstead's Measures of complexity, McCabe Cyclomatic Complexity.

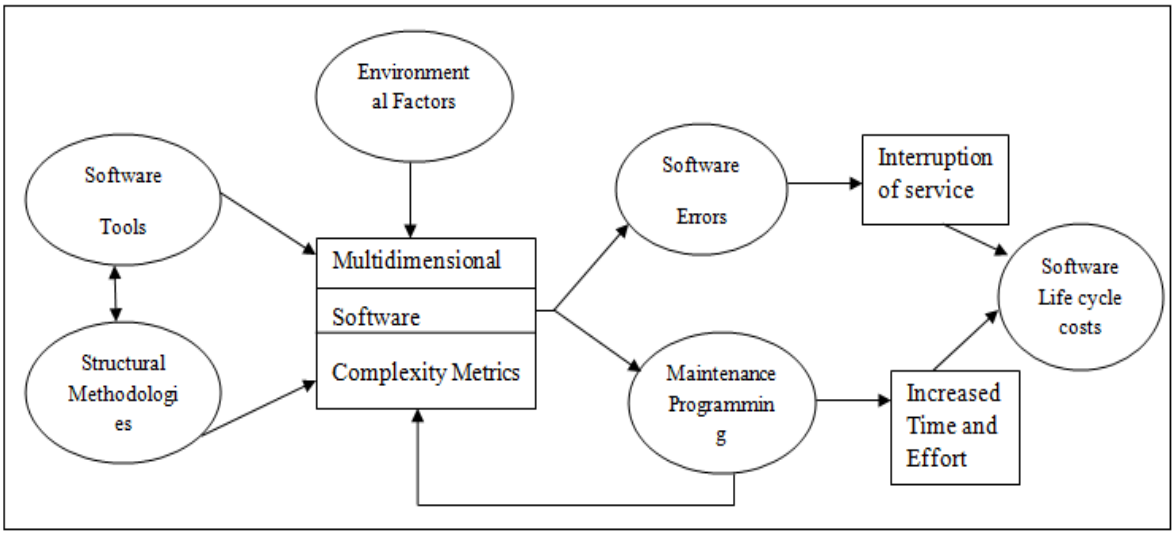


Fig.1.2 Framework for Evaluating Economic Impacts of Software Complexity [8].

The relationship between software complexity and software cost is shown in fig.1.2. Higher will be the complexity, higher will be the cost and difficult to maintain the software. Various tools are used on the left side to control and reduce the software complexity [8]. These three metrics LOC, Halstead's and cyclomatic complexity are

used to measure the complexity of software. There is a problem with LOC metric. This metric is used to count the number of lines in the program's source code. It is calculated at the end of the application completion, ignores the complexity from decision statements and consider the complexity of each code line same. So, to remove this difficulty a new metric called Halstead's complexity metric was introduced. This metric is used to measure the complexity of program or module's directly from the program's source code but there was also some problem with this metric. It depends upon completed code means always calculated at the end of application completion but in includes the complexity from data flow. To overcome the above problem, McCabe's Cyclomatic Complexity was introduced which can be computed before the application completion or can be calculated at the early stage of software development life cycle as compared to Halstead's metric. This section focuses on Cyclomatic complexity (or conditional complexity) which was developed by T. J. McCabe in 1976.

It finds the number of linearly independent paths in a program's source code**.** It is one of the metric used to measure the complexity or number of independent paths in a program. For good quality software and error-prone software, it is very necessary to measure the complexity of program. McCabe Cyclomatic Complexity is used to measure the structural complexity of modules. More the Cyclomatic Complexity number then more will be the chances of error in the program and hence extra effort will be required to correct the errors or faults [9].

Here, focus is on software complexity and many of things are collected about a software and project during its development.

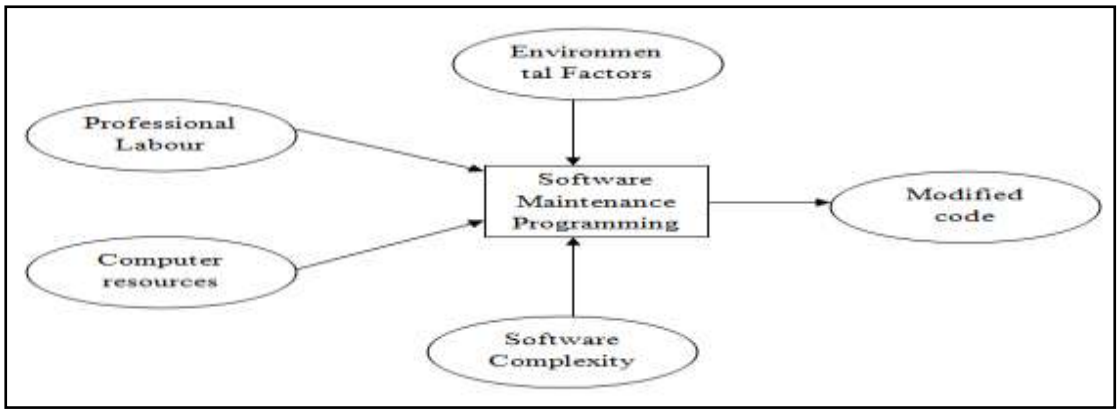


Fig.1.3 Model of Software Maintenance Programming Code [8].

Fig.1.3. Shown above considering important factors, which must be considered or collected during software development.

So, one popular “Metric” plug-in has been used widely which provides many features like total number of attributes, classes, child classes, interfaces, methods, total number McCabe Cyclomatic Complexity of a program of a project etc.

These are the following requirements for using the “Metric” plug-in:

1. There should be a Java or Java Browsing perspective.
2. After that, select a project and enable the metric from the context menu and to enable a metric firstly go to windows option then show view option and in last navigate to metric view.
3. In last perform a rebuild on the project [10].

There are three methods for finding Cyclomatic complexity.

1. Draw the control flow diagram and then count the number of edges and number of nodes from that control flow diagram. After that apply the formula

   Cyclomatic complexity = #edges - #nodes + 2

2. Number of decision statements in a program of project plus one.
3. Calculate number of regions form the control flow diagram plus one.

1.4 Thesis Outline

The thesis has been organized into six chapters namely: Introduction, Literature Review, Problem statement, Implementation, Snapshots of the application and Conclusion. The Chapter 1 gives the brief introduction about the domain of the topic which is the software engineering discuses about software definition, software development steps, briefly discusses about software complexity and various metric techniques used to measure the software complexity. Along with the domain, the main topic of the thesis which is the cyclomatic complexity has been discussed telling about the importance of the cyclomatic complexity concept and why it is so growing. Chapter 2 is the literature review which contains the main research and study part of the thesis; it contains the different types of software complexity metrics and includes comparison between different-2 complexities metrics, different methods used to

measure cyclomatic complexity number and importance of cyclomatic complexity number, Correlation of cyclomatic complexity with these two metrics, with error density and failure rate also get discussed in this chapter.

Chapter 3 explains the problem statement. Chapter 4 gives the full implementation which includes proposed algorithm related to problem, concept of coupling, different types of coupling also discussed. Chapter 5 includes demonstration of the project work through snapshots, tabular form and graphs. Also contains the brief introduction of the platform. Lastly the chapter 6 includes the conclusion and the future scope of the research work performed.

CHAPTER 2

LITERATURE REVIEW

2.1 Source lines of code

Source line of code (SLOC) is one of the simplest, oldest and widely used metric to measure the size of a program. This metric is used to estimate man-hours for a project. It measures the program size by counting the total of lines of the program code. LOC is usually defined as: Lines of Code (LOC), it counts each and every line in the program including blank lines and comments.

Kilo Lines of Code (KLOC) = LOC / 1000.

Effective Lines of Code (ELOC), counts line of code and excludes parenthesis, blanks and comments.

Once LOC is calculated, we can measure the following attributes:

1. Productivity of software is measured as LOC divided by Persons months.
2. Quality of software or project is number of defects in given software or project divided by LOC and cost of software is also directly related to LOC.

Some recommendations for LOC:

1. File length should be 4 to 400 of total lines of code.
2. Function length should be 4 to 40 of total lines of code.
3. Comments should be in the range of 30 to 75 percent of total line of codes [11].

2.1.1 Advantages

1. LOC is mostly used and measured after project completion.
2. It is language independent and has been proved to be useful as a predicator of program effort.

2.1.2 Disadvantages

1. LOC can be counted only when the application is done and it is always counted at the end of the life cycle. It is very difficult or not possible to count the lines of code at the stage of early life cycle.

2. Ignores the complexity from the decisional statements or conditional statements if present in the program and also ignores the complexity from the data complexity.

So, to remove this difficulty a new metric Maurice Halstead in 1977 was introduced. This metric is known as Halstead software science or as Halstead metric.

2.2 Halstead's Measures of Complexity

Halstead's Measures of Complexity is mainly used to measure the error rate of a program. Halstead classified a program P as a collection of tokens, either operands or operators. Halstead metrics are based on the following indices:

1. op1 - distinct total # of operators in a program.
2. op2 - distinct total of operands in a program.
3. OP1 - total # of operators in a program.
4. OP2 - total # of operands in a program.

Operators and Operands identification based on the programming language used.

```
Void sort (int *a, int n)
{
    int i, j ,k;
    if (n < 2)
    return;
    for (i=0; i<n-1; i++)
{
    for (j=i+1; j<n; j++)
{
    If (a [i] >a [j])
{
     k = a [i];
    a [i] = a [j];
    a [j] = k;
}
```

}
}
}

Now, identify the operators and operands from this program and then after the identification of these unique operands and operators we can identify the various attributes for particular software like program length, difficulty level etc.

Table 2.1 Calculation of operators and operands of program [11].

Operators	Occurrences	Operands	Occurrences
<	3	0	1
-	5	1	2
>	1	2	1
_	1	A	6
,	2	I	8
;	9	J	7
(	4	N	3
)	4	K	3
[]	6		
For	2		
If	2		
Int	1		
Return	1		
{	3		
}	3		
+	1		
++	2		

Table 2.2 Calculation of Halstead's Indices [11].

	Total	Unique
Operators	OP1=50	op1=17
Operands	OP2=30	op2=7

Based on these, Halstead defines a number of attributes of software.

This metric is use to measure error rate, length, difficulty level etc. for a software development [12].

$$ESL = op1 \log_2(op1) + op2 \log_2(op2) \quad (1)$$

$$SL = OP1 + OP2 \quad (2)$$

$$SV = op1 + op2 \quad (3)$$

$$VOL = SL * \log_2(n) \quad (4)$$

$$V^* = (op1 OP2 / 2op2)(OP1+OP2) \log_2(op1+op2) \quad (5)$$

$$LVL = V^*/VOL = (2/ op1)*(op2/ OP2) \quad (6)$$

$$DL = VOL/V^* = (op1/ 2) * (OP2/op2) \quad (7)$$

$$PE = VOL * DL \quad (8)$$

$$EE = VOL/S^* \quad (9)$$

$$PT = PE/18 \quad (10)$$

Where,

ESL: Expected software length

SL: Software length

SV: Software vocabulary

VOL: Volume

LVL: Level

DL: Difficulty Level

PE: Programming Effort

EE: Error Estimate

PT: Programming Time

V*: software ideal volume.

S* is the programmer ability's and Halstead's set this value to be 3000.

Now, Halstead defines a number of attributes of software like Program vocabulary, Program length, Program volume, Difficulty level, program level, Programming effort etc.

1. **Program vocabulary:** it counts the number of distinct operands and unique operators.
2. **Program length:** it counts the total number of operators and total operands.
3. **Program Volume:** is another measure of program size.
4. **Difficulty Level:** This metric is proportional to the total # of distinct operators (op1), operands (op2) and also total of operands (OP2) in the program. The use of same operands in the program several times, causes errors.
5. **Program Level:** Program level is the inverse of the level of difficulty.
6. **Programming Effort:** Programming Effort defined as: E =VOL*DL.

2.2.1 Advantages

1. Halstead's metric is used to measure the overall quality of a program.
2. It does not require deep knowledge of programming and very simple to calculate.
3. Halstead's metric can also be used to measure rate of errors and effort.
4. Used to calculate the complexity from the data flow of software.

2.2.2 Disadvantages

1. In Halstead's metric it is very difficult to separate operators and operands from the source's code [13].
2. Also, Inheritance, Interaction between modules cannot be measured with Halstead's metric method.
3. It also ignores the complexity from the decision statements like if, loops etc.

2.3 Importance of Cyclomatic Complexity

Cyclomatic complexity is one of the best metric to measure the complexity of any software or software module. It is used to measure the complexity of software via the number of independent path or number of independent flows through the graph. It is direct indicator of software cost and quality because these two parameters are directly related to software complexity. The greater the complexity greater will be the fault in software and resulting in higher cost. Much effort will be gone into identifying techniques and metrics to 'measure' the complexity of software and software

modules. Module having higher complexity considered low cohesion or lower complexity modules considered as high cohesion. Cohesion defined as the extent to which modules of a program's source code are bind together. Cohesion (within the module) should be high and coupling (between modules) should be low. It is considered that highly cohesive module is easy to understand and logical implementation of that module is clear and also directly related to quality [14]. Different types of cohesion types are possible between modules.

1. **Coincidental cohesion (worst):** When software or module parts are grouped arbitrary then relation between grouped parts comes under coincidental cohesion.
2. **Logical cohesion:** When software or module parts are divides according to their logical behavior and those categorized parts does the same work to meet the objective and they can be different according to their nature like keyboard and mouse.
3. **Temporal cohesion:** According to processing when modules are grouped then temporal cohesion occurs.
4. **Procedural cohesion:** When work is done according to some sequence and then execution occurs like firstly open the file then read operation is performed**.**
5. **Communicational/informational cohesion:** when software or module part works on same data then communicational cohesion occurs.
6. **Sequential cohesion:** when output of module is the input for another module. For example lexical analyzer.
7. **Functional cohesion (best):** when module part works together to do the same work then functional cohesion occurs.

Along with cyclomatic complexity, there are also other methods or metrics like source line of code, Halstead's Measures of Complexity etc. but there are some problems with these metrics that's why the concept of cyclomatic complexity comes.

2.4 Correlation of cyclomatic complexity concept with LOC and Halstead's measure of complexity

This metric has a strong correlation with LOC [15] and also along with Halstead's measure of complexity. Instead there is no relation between control path and number of operators and operands but still with the increase of control path, number of operators and operands also get increased. So, this shows a correlation between Halstead's and Cyclomatic complexity metric [16].

Cyclomatic complexity metric does not consider the complexity from the data flow of software. Example if there are 1000 lines of code in any software and there is no conditional statements in the code then cyclomatic complexity metric calculates the complexity of that software as one. One more problem with this metric is that it considers the complexity of two statements having while and if as same.

2.5 Cyclomatic complexity

LOC metric is used to measure the size of a program and McCabe's Cyclomatic Complexity is used to measure the complexity of program and there is a difference between size and complexity. For example if there are 100 lines of code but there is no branching(like if, for etc.) then in this case complexity of a program is one but line of code are 100 and there can also be a possibility of 100 lines of code has 90 complexity. So, both can't be used for same purposes [17].

Cyclomatic complexity was introduced by McCabe and used to measure the complexity of a piece of code that code was written in FORTRAN language. FORTRAN language is a linear programming language in which there are no classes and functions [18].

There was no concept of object oriented and structured languages at the time when Cyclomatic complexity was introduced. Because of ignorance of this concept in Cyclomatic complexity it was not considered as a strong metric to measure the complexity of programming language like OOPs. Physical size i.e. LOC metric metric before this technique was not considered adequate because if there are 40 or 50 line of code consisting of various consecutive conditional statements like "while", "if" then it might be possible, that program code consists of million distinct control paths, only a small part of which would probably ever be tested [19].

McCabe's Complexity metric is a software quality metric and higher the Cyclomatic complexity number, the more complex the code will be. Cyclomatic Complexity is also used to measure the structural complexity of a module.

McCabe's technique is also called conditional complexity [20] because it considers only decision control statements such as "If else" and "loops" etc. McCabe also used Cyclomatic complexity for testing of code and the numbers of test cases were equal to Cyclomatic Complexity.

2.5.1 Advantages

1. Can be computed at early stage of life cycle.
2. Easy to predict the cost, testing effort from the McCabe cyclomatic number.
3. Easy to apply and calculate.
4. Predicts the software complexity from the decisional statements.

2.5.2 Disadvantages

1. Cyclomatic complexity metric does not consider the complexity from the data flow of software.
2. It considers the complexity of two statements having while and if as same.
3. Also, Inheritance, Interaction between modules cannot be measured with this method.
4. Statements with if blocks or with if/else block has considered the same complexity.

```
Like, if (condition)          if (condition)
{                             {
}                             }
                              Else {
                                    }
```

Complexity of both source code's is same i.e. 2.

5. Complexities of nested statements and without nested statements are considered as same.

```
Like, if (c1) and (c2)        if (c1)
{                             {
Statement1                    if (c2)
}                             {
```

Else {
}

Statement1
}
Else {
}

Complexities of these two considered as same i.e. 3. But, it's clear that complexities of nested statements are difficult to understand as compared to without nested statement [21].

Table 2.3 Comparison of Software Complexity Metrics

S. No	Parameters	Software Complexity Metrics		
		LOC	**Halstead's**	**McCabe's Cyclomatic Complexity**
1	**Approach Used**	Uses physical length of the code.	Uses the count of unique operators and operands.	Calculates the number of independent paths from control flow graph.
2	**Software Life Cycle Phase**	It can be calculated either at the coding stage or after the end of the complete life cycle phase.	It can be calculated only at the end of the complete life cycle phase.	It can be calculated at the design or code phase of the life cycle.
3	**Bug Density**	Concave relationship	Forecasts the bug density.	Highly related
4	**Base used for calculation**	Source Code	Source Code	Logic Structure
5	**Language**	Language	Language	Language

		independent	dependent	independent
6	**Usability**	Easy	Medium	Medium
7	**Data and Control Statements**	Ignores the complexity generated by the decision statements.	Considers the complexity due to data but ignores the complexity due to decision statements.	Ignores the complexity due to data but considers the complexity due to decision statements.
8	**Theory Base**	No	No	Yes
9	**Additional Uses**	Productivity and man effort can also be calculated.	Error rate, vocabulary, code length, difficulty level, volume, effort, time can be calculated.	Risks associated, effort, relative complexities can be calculated.
10	**Popularity**	Narrow	Wide	Wide

2.6 Three methods for Cyclomatic Complexity

McCabe's Cyclomatic number used to study program complexity may be applied. There are 3 methods to calculate the Cyclomatic Complexity number from a flow diagram.

1. Number of edges – number of nodes + 2 [22].
2. Number of binary decisions + 1[23].
3. Number of closed regions + 1[23].

1. First Method

In first technique creates a control flow graph of a program's source code and then measures all linearly independent path of that program's code. This control flow graph consists of nodes which are connected by edges and then complexity is measured through this control flo

Cyclomatic complexity measured by:

CC (Cyclomatic complexity) = E – N + 2P

Where, E is the total of edges present in graph, and represents the flow of control between nodes.

N is the total of nodes represent expressions and statements.

And number of connected component represented by P.

So, in first method, cyclomatic complexity is measured by the flow graph. Now, consider a program having seventeen lines of code having corresponding start and end statement. This program is basically for calculating the power of any number.

```
1     begin
2     int a, b, pow;
3     float c;
4     input(a, b);
5     if (b < 0)
6     pow = -b;
7     else
8     pow = b;
9     c = 1;
10    while ( pow!=0) {
11    c=c*a;
12    pow = pow -1;
13    }
14    If ( b<0)
15    c = 1/c;
16    Output( c );
```

17 End

This is the program for a raised to power b.

Firstly, create a blocks from this program and that represented in tabular form consists of four column and after that flow graph is generated from this tabular form. The graph represents the flow of source code.

There are total nine nodes and 13 edges in this control flow graph.

Control flow graph actually represents the logic structure of a program's source code or module which has only single entry point and exit point.

Here node 1 represents if statement in a program and if condition is true then go to node number 2 else go to node number.

Table 2.4 Basic Blocks

Block	Lines	Entry Point	Exit Point
1	2,3,4,5	1	5
2	6	6	6
3	8	8	8
4	9	9	9
5	10	10	10
6	11,12	11	12
7	14	14	14
8	15	15	15
9	16	16	16

Control Flow Graph is very easy to understand and always gives useful results. In CFG there is a node labelled Start that has no incoming edge, and another node labelled End that has no outgoing edge. From that graph now calculate the number of edges and nodes and put in formula.

I.e. cyclomatic complexity number = #edges - #nodes +2P.

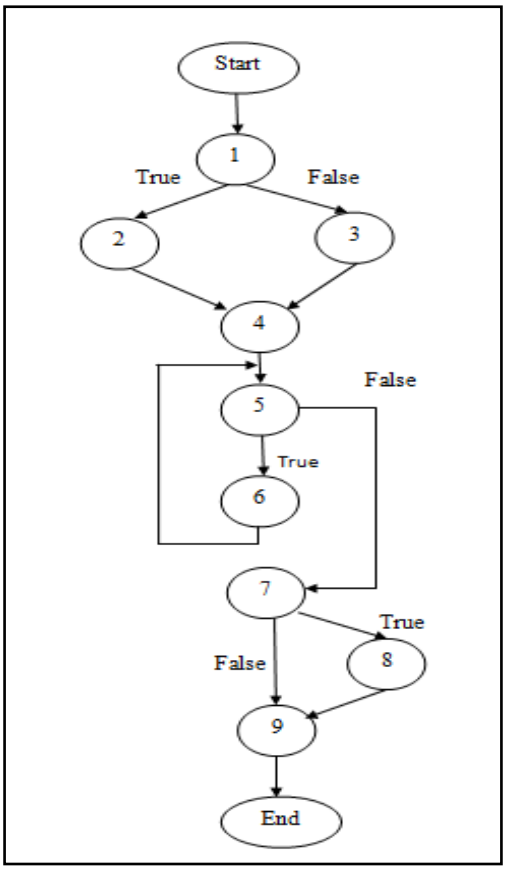


Fig.2.1 Control Flow Graph

From the above program #edges =13, #nodes = 9

So, cyclomatic complexity number = 13-11+2 = 4.

Table 2.5 Number of Independent paths calculation.

Paths	Independent path
Path 1	Start,1,3,4,5,7,8,9,End
Path 2	Start,1,2,4,5,7,9,End
Path 3	Start,1,3,4,5,6,5,7,8,9,End
Path 4	Start,1,2,4,5,6,5,7,8,9,End

There are four independent paths in this graph. So, from independent path can easily judge the cyclomatic complexity number. Cyclomatic complexity number is equal to the number of independent paths in a graph.

Independent path is a path which cannot be traversed again and which process new set of statements each time.

2. Second Method

In second method McCabe Cyclomatic Complexity is calculated by determining the number of decision statements which are caused by conditional statements in a program and plus one.

Cyclomatic Complexity = number of decision statements + 1.

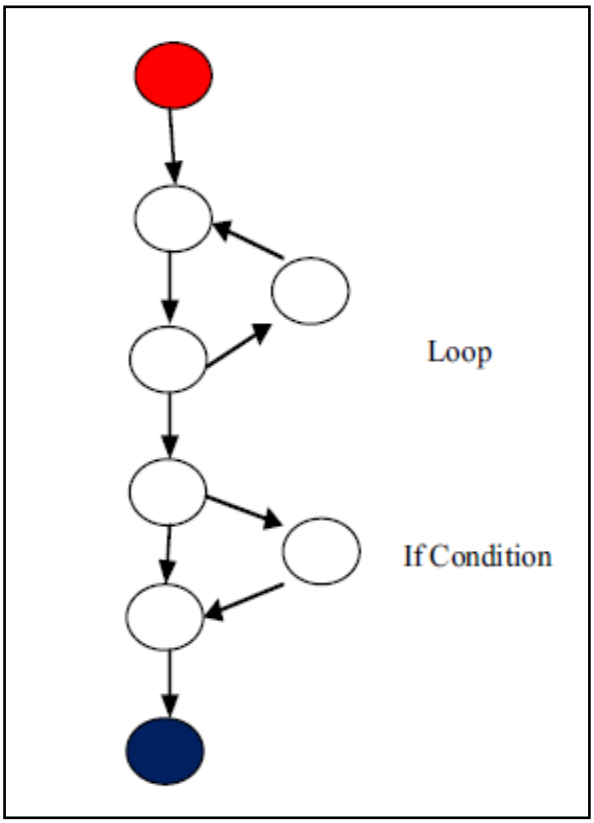


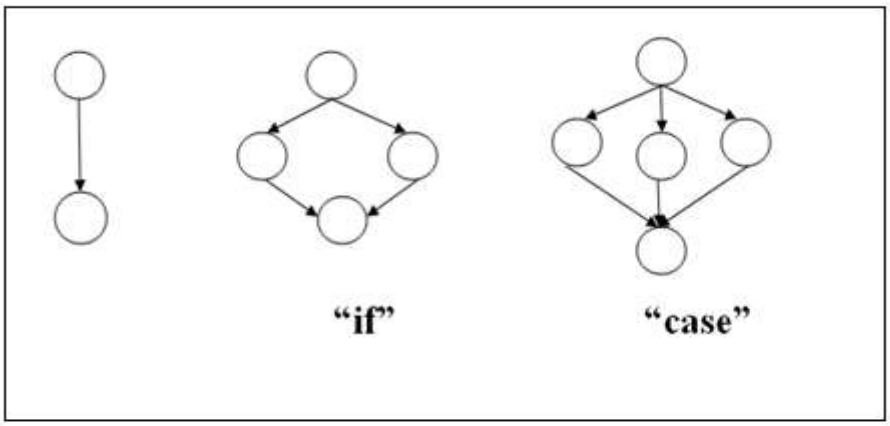
Fig.2.2 Decision Control graph [18].

There are some basic rules that can be used to measure McCabe's cyclomatic complexity number.

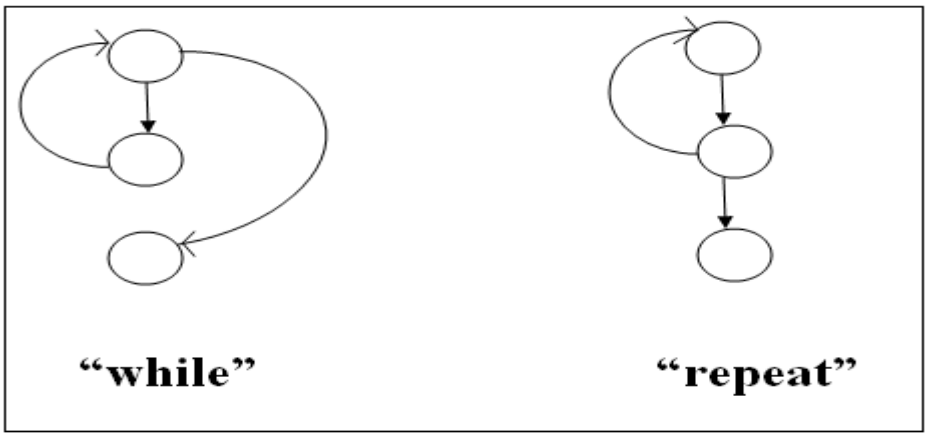
1. Calculate the number of if/then, else if but do not count the else statements in the program.
2. Find the switch statement and count the total of the cases in the program but do not count the default in the program.
3. Calculate all the loops like for, while and do-while statements and also all the try/catch statements in the program.
4. Count conditional operator && and || operator and also ternary operators like?: from the expression.

Now, add one to the numbers from the previous step numbers and that will be the final cyclomatic complexity number of the program. So, in the above program number of decisional statements are three (two if statements and one while statement) and then add one for final cyclomatic complexity number.

So, final cyclomatic complexity number for above program according to second method is = 3+1=4.



Sequence **Selection**



Iteration

Fig.2.5 Structured programming primitives [16].

3. Third Method

Cyclomatic complexity = Number of enclosed areas + 1

So, in this also firstly draw the control flow diagram and then calculate the number of closed regions in control flow diagram. Here, number of closed regions are 4.

So, cyclomatic complexity is= 3+1=4.

From all three methods we get the same Cyclomatic complexity number.

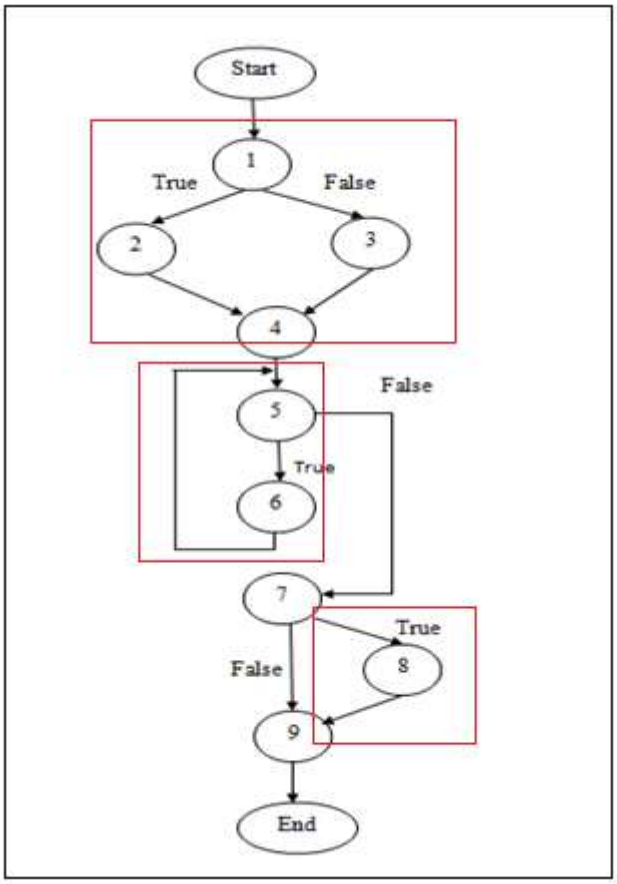


Fig.2.6 Regions calculation from the flow graph.

2.7 Multilevel testing based on Cyclomatic Complexity

Complexity is found at all the phases of the software life cycle. If we find high complexity processes then reconstructing the processes can make them easily testable.

Structured testing is a testing that use systematic approach to generate test cases so that each test case uniquely identify all the errors. Structured testing uses approach in which design phase is divided into segments or we called it tiers. In this there are four tiers [24]. So, in tier 0 divide the program in to subprograms and allocate the

requirements to each subprograms. In tire 1 divide the subprogram in to modules and in tier 2 modules in to procedures and completed details are assigned to the procedures.

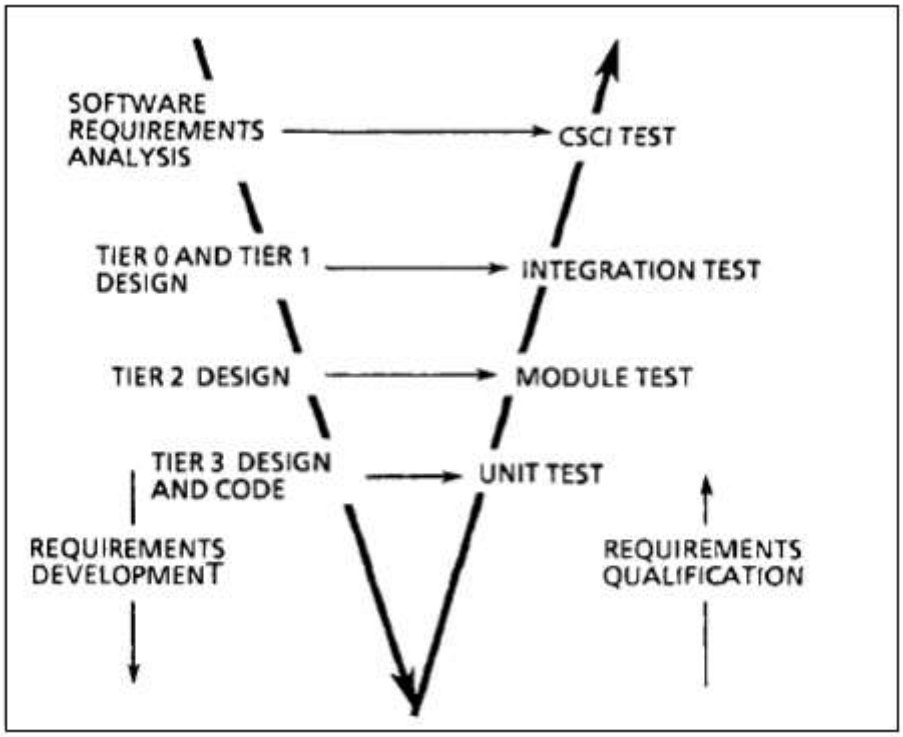


Fig.2.7 Multi-level testing based on Cyclomatic Complexity [24].

After that unit testing is done at the tier 3 and in this each procedure is considered as node. So, calculate the cyclomatic number for each procedure and if this cyclomatic number is greater than 10 then reconstruct the procedure again and then we do unit testing on a procedure.

Unit testing test each procedure. After that module testing is done on tier 2 and again we calculate a cyclomatic number for tier 3 and check whether it is greater than 10 or not. Basically, we are generating a test cases and number of test cases equal to the cyclomatic number at each tier and testers apply these test cases during testing.

2.8 The Significance of the McCabe Number

According to some facts if programs with McCabe Cyclomatic complexity number greater than 10 then program has higher probability of containing errors and defects and it is very difficult to understand the program. In this case more number of test cases will be required to execute the paths in the program. Means higher the

Cyclomatic number higher will be the error rate, and more maintenance or refactoring the code will be required.

Software Engineering Institute, categorized risk evaluation corresponding to Cyclomatic Complexity as follows:

Table 2.6 Risk Evaluation corresponding to cyclomatic complexity number [25].

Cyclomatic Complexity	Risk Evaluation...
1-10	Software considered as risk free software.
11-20	Software considered as risk of moderate risk.
21-50	Software considered as of high risk.
51 and greater	Considered as of highly risk and even not testable software.

From all three methods we get the same Cyclomatic complexity number. This calculated cyclomatic complexity number should be in range of 1 to 10 and only then software is considered as risk free software. If the same lies in the range of 10-20 then it is considered as a target of moderate risk. 30-40 range of cyclomatic number makes module highly risky and the range exceeding 40 exempt it from the candidate of testing [25]. Based upon some data it has been proven that higher will be the cyclomatic complexity number lower will be the quality of software.

2.8.1 Correlation of High complexity with failure

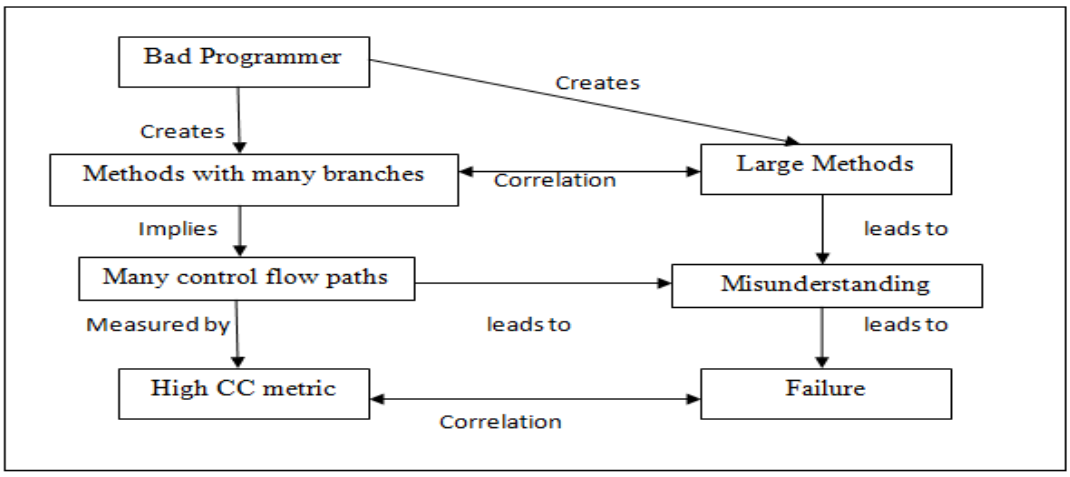


Fig.2.8 Two comparable explanations for correlation of high CC with failure [26].

Higher software complexity leads to failure in software. Two cases can be possible for that.

1. Bad programmer writes program having large methods leads to failure.
2. Programmer writes methods with many jump statements and branch statements having many control paths also lead to failure.

2.9 Anatomy of cyclomatic complexity

In this section the focus is on the cyclomatic complexity. Why this is preferred over two metrics LOC and Halstead's metric, Different methods to calculate the Cyclomatic Complexity number and what will be the effect of this cyclomatic number on other parameters.

All three metrics are used to measure the software complexity but only cyclomatic complexity metric measures the complexity of software from conditional statements. Cyclomatic complexity metric uses three methods to calculate the cyclomatic complexity. In first method control flow graph is generated from the source code and then calculate the number of nodes and edges from graph.

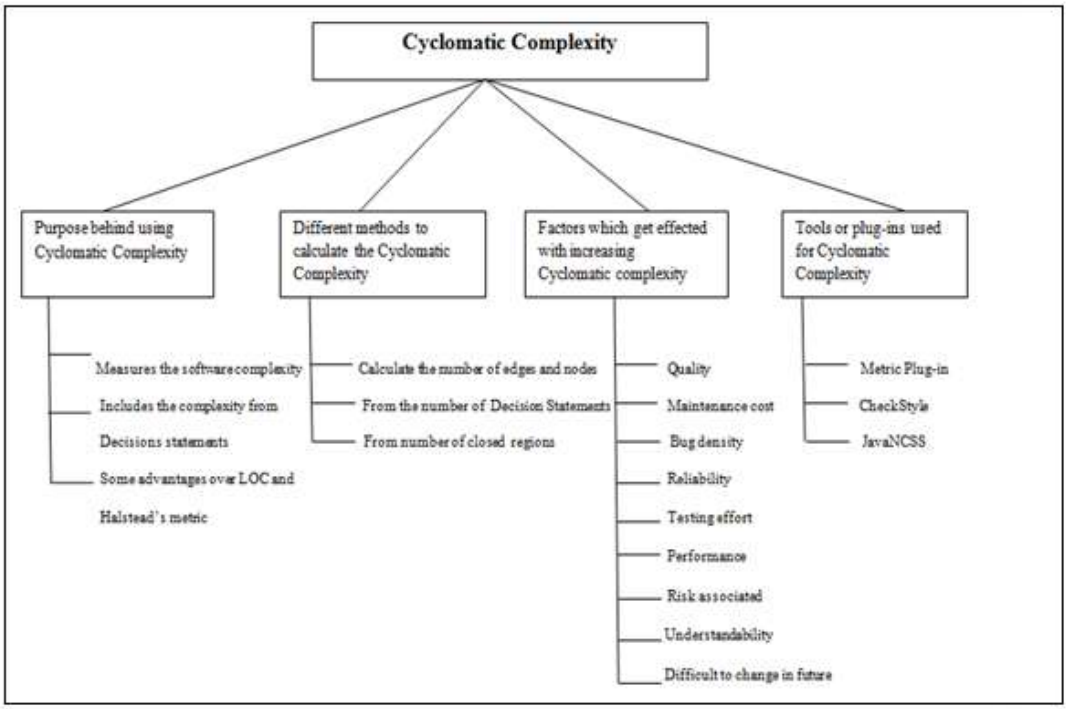


Fig.2.9 Cyclomatic complexity metric

Second method uses or calculates the number of decision statements directly from source code and in third method number of regions are calculated from the control flow graph. After that if this calculated cyclomatic number is greater than 10 then there will be a chance of more errors in the software then more testing effort will be required to find these errors. So, reliability, performance, maintenance cost, quality and other factors will get affected. Various java plug-in are used to measure the cyclomatic complexity. One of the plug-in names is “Metric” plug-in.

To use Metric plug-in

Firstly Metric plug-in requires Eclipse 3.1

And after that these are three conditions to use the metric plug-in.

1. There should be a Java or Java Browsing perspective.
2. After that, select a project and enable the metric from the context menu and to enable a metric firstly go to windows option then show view option and in last navigate to metric view.
3. In last perform a rebuild on the project.

There is two or three plug-ins for java to find the Cyclomatic complexity and they all are calculating the intra-modular Cyclomatic complexity of a program or project.

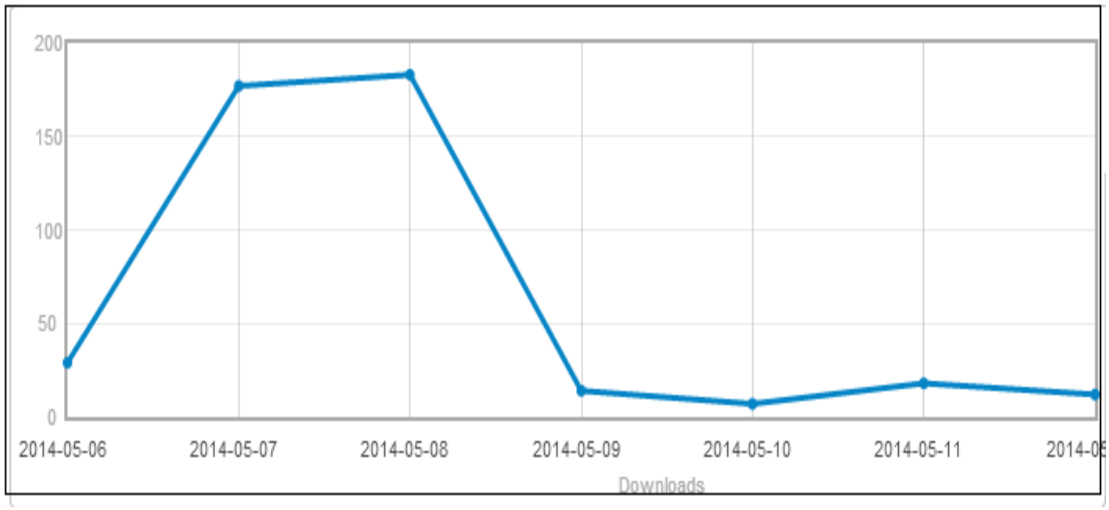


Fig.2.10 Graph for Metric Plug-in [27].

But in actual there is also need to consider inter-modular Cyclomatic complexity. Interaction between modules is not possible in cyclomatic complexity. Now, the table for 440 downloads for Metric plug-in per week.

Table 2.7 440 downloads for "Metric" Plug-in per week [27].

Date	Downloads
2014-05-06	29
2014-05-07	177
2014-05-08	183
2014-05-09	14
2014-05-10	7
2014-05-11	18
2014-05-12	12
Total	440

CHAPTER 3

PROBLEM STATEMENT

3.1 Problem Statement

McCabe cyclomatic complexity is not considering or measuring the exact software complexity means if there is interaction between two or three object classes in software then it does not calculates that complexity. There was no concept of object oriented and structured languages at the time when Cyclomatic complexity was introduced. Because of ignorance of this concept in Cyclomatic complexity it was not considered as a strong metric to measure the complexity of programming language like OOPs. There is a need to consider that object coupling complexity in McCabe cyclomatic complexity. It does not calculate the exact software complexity. There is a need to consider other factors to calculate the exact complexity of software so that an accurate prediction and assumption can be made from the exact complexity number.

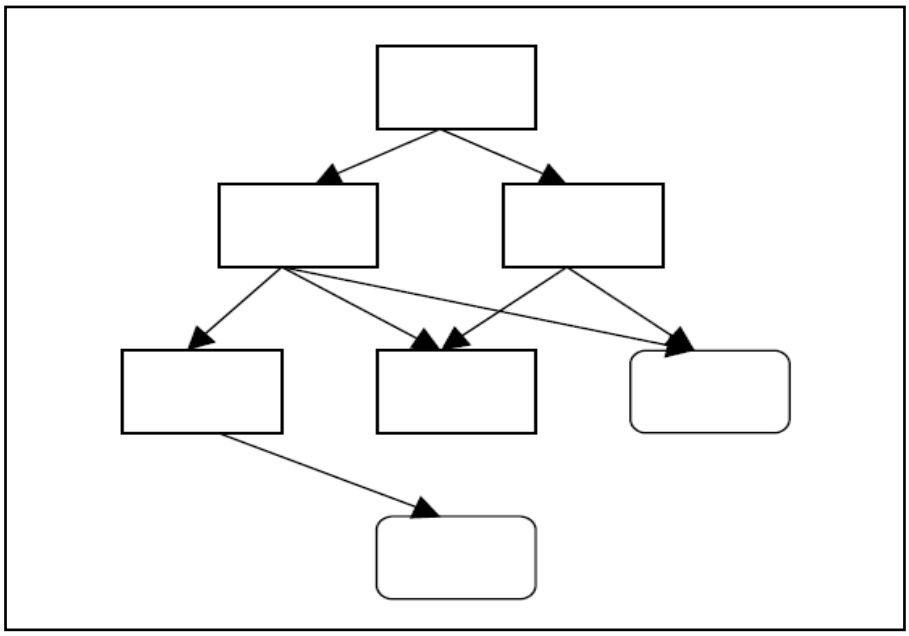


Fig.3.1 Inter-Modular Relation

And also, McCabe Cyclomatic Complexity does not consider inter-modular complexity means interaction between the different modules. If function f1 that has a complexity a and function f1 calls another function with complexity b, then in this case the Cyclomatic complexity of this function f1 will be a. But in actual invoked

function complexity b higher than of function f1 complexity but McCabe has ignored the complexity of the invoked function. So that's why there is a need to consider the concept of coupling in cyclomatic concept.

Cyclomatic number does not consider any complexity when a code interacts with other code e.g. Class A is calling some function of class B. For these interactions then there is a another factor i.e. CBO(complexity between objects).CBO is nothing but just the number of unique references a class has to other external classes by any means say it can inherit other class or some of its method has a return type of some external class etc . Other possibilities will be discussed in the algorithm for calculating CBO e.g. Class A inherits Class B and it has some function whose return type is Class C. In that case CBO for Class A will be 2.

```
class B {
    public void f2() {
       System.out.println("statement");
           }
       }
public class CM {
    public static void main(String args[]){
            B obj3= new B();
                obj3.f2();
                }
                }
```

Now, simplest example to show the interaction between two object classes. This example consists of two classes and coupling between these two object classes are one.

CHAPTER 4

PROPOSED SOLUTION

Firstly there is a need to consider the concept of coupling and its importance and its effects on various quality attributes.

4.1 Coupling

Coupling should be taken care of because it measures the degree of interdependence between two modules or object classes means change in one object class affects another object classes [28] . Two or more classes can be either loosely coupled or highly coupled based on the number of ways they are coupled. If two classes are coupled in more than one way then they are highly coupled classes else loosely coupled.

If coupling is not considered while calculating the complexity it will lead to some pitfalls in the quality factors of the program. The factors that are affected by this includes

1. Reliability [29]
2. Reusability
3. Understandability
4. Cost
5. Testing effort
6. Fault prediction
7. Quality
8. Modifiability
9. Maintainability.

If there is strong connection between two modules or classes then failure in one cause a failure in another component and for that reason programmer also requires more understanability of software. Testing effort will also get increased due to fault in one component which is connected to another component because programmer requires checking the other component. So, these quality factors like

reliability, reusability, quality, testing effort and understandability etc. will get compromised due to interaction between classes or modules in a class [30].

4.2 Types of interaction

Two types of interaction can be possible

1. Another one is between the modules with in the class i.e. intra class interaction.
2. Between the object classes [31].

4.2.1 Intra class interaction

Intra class type of interaction occurs between modules (like functions) means with in a class. Some different types of coupling are:

1. **Data coupling:** Two modules are data coupled if one passes information or data as argument to another. For example parameter or passing an integer to a function that computes a square root.
2. **Stamp coupling:** If two modules communicate via composite data structure and use only a part of it to perform their actions or functions.
3. **Control coupling:** If one module passes a value to another which is used to control the sequence execution of another.
4. **Common coupling:** Common coupling occurs when sharing of same global data between two modules. For e.g. a global variable. Common coupling is also known as global coupling. Changing that global variable implies changing all the modules using it.
5. **Content coupling (high):** Two modules are content coupled if one of module either relies or modifies the content of other module. For e.g. accessing the local data of another module.

4.2.2 Interaction between object classes

Coupling can occur among object classes through different methods like Field accesses, through methods calls, Inheritance, Arguments, Return types, Exception, instruction type.

1. Inheritance related coupling: Inheritance related coupling occurs between two classes through two most commonly keywords i.e. extends and implements and also are used to show the whether one object IS-A type of another

A class uses extend keyword to extend the parent classes and use implements keyword to implement the interfaces and a class can also implements more than one interface.

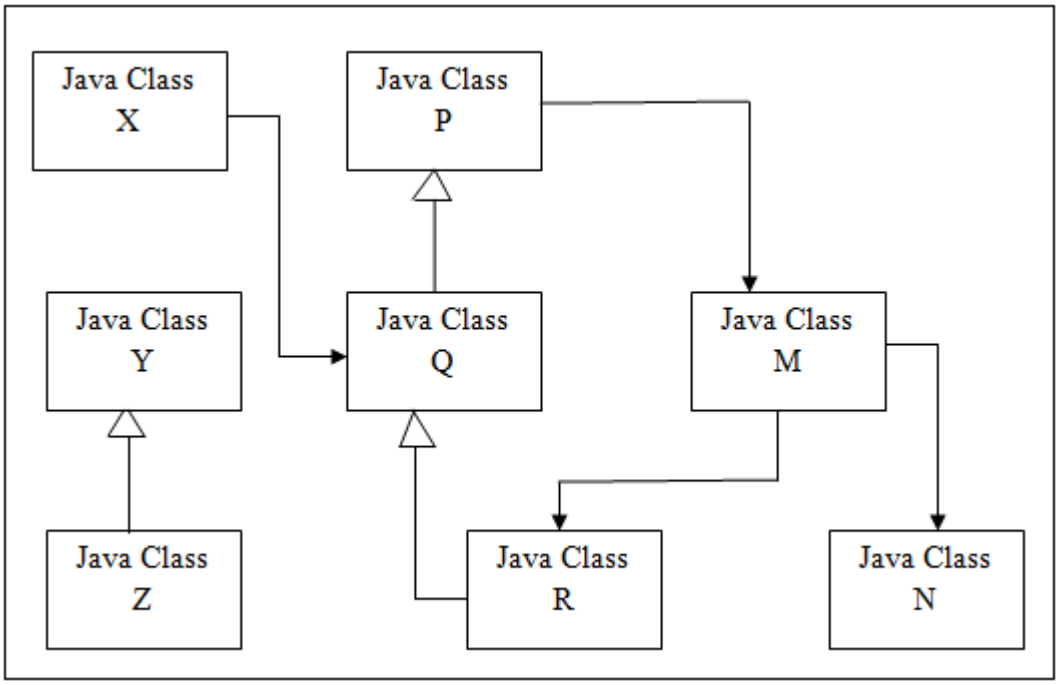


Fig. 4.1 CBO of java classes [32]

Now, Pseudo code for how two classes are coupled through inheritance.

```
Class A {

    Public void f1 ()
    {
    System.out.println("statement1");
    }
    Class B extends A
    {
    System.out.println("statement2");
```

```
}
```

Interaction between classes increases due to interaction and that also increases the complexity of software [33].

2. Argument type coupling: Coupling can be possible between two classes via passing argument. Means if there are two classes like A and B then passing the object of class B as argument to class A function definition incurs the interaction between two classes. Coupling occurs when passing a message from one to another class through argument or parameters which contained in message [34].

3. Through Constructor: And through the constructor call coupling between two classes will get increased.

1. X a = new X (); // implicit constructor
2. a.f1 (); // explicit, through an object reference
3. X.m(); // explicit constructor.

B these three ways coupling between classes gets increased.

4. Through return types: Take an example of program that will explain the case of coupling through argument passing through return types.

```
class A {
    public void f1() {
        B ob2=new B ();
        ob2.f2 ();
        System.out.print ("s1");
            }
        }
class B extends A{  //inheritance
    public void f2() {
```

```
        System.out.print ("s2");
}}
public class C {
 public B getObject(){
              B ob2 = new B ();
              return ob2;   //return type
                }
            public void f3(A ob2){
              ob2.f1 (); //local variable or argument
}
            public static void main(String args[]){
          if (condition1) {
            System.out.print ("Statement1");
          } else if (condition2) {
            System.out.print ("Statement2");
          } else if (condition3) {
            System.out.print ("Statement4");
          } else {
            System.out.print ("Statement4");
                        }
                    }
                }
```

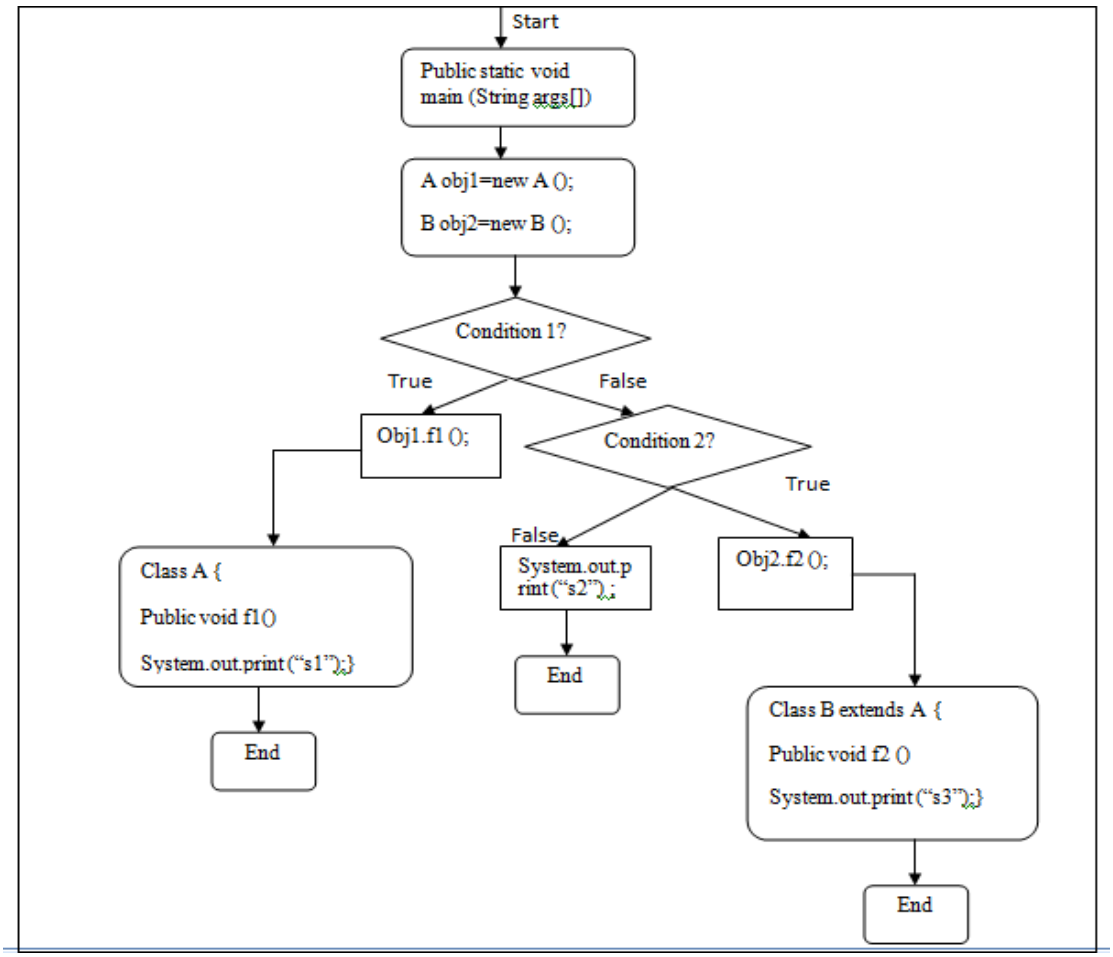


Fig.4.2 Flow graph corresponding to source's code

Control flow graph drawn for above source code represents flow of control between classes and can be easily understand the concept of coupling and interaction between object classes from this graph.

The graph starts from main statement present in graph because program execution always starts from main and after that according to statements flow is shown. This program trying to include some rules to show the interaction through constructor call, through local variable or arguments. Now purpose is showing firstly the cyclomatic complexity according to concept of decisional statements. For that an algorithm is written and then to calculate the total complexity (due to conditional statements and due to coupling between object classes).

4.3 Algorithm for McCabe's Cyclomatic Complexity

Find McCabe's cyclomatic complexity

Input: Java File

Output: Cyclomatic complexity

/*keyword is an array of all the decision making keywords which affect cyclomatic complexity, cyclomatic is the variable that holds the cyclomatic complexity

Initialize it with 0*/

1. Read the input file

2. For each java file Read it with bufferedreader line by line

While (buffer! = NULL)

For each line divide it into tokens with StringTokenizer

if (token equals keyword[i])

cyclomatic ++;

break;

endif

endfor

endwhile

endfor

3. Final cyclomatic complexity will be cyclomatic+1.

This algorithm is written on basis of cyclomatic complexity concept. Based on the concept of decisional statements like if/else, try/catch block, loops (i.e. for, while, do-while) and operators like &&, || and? : Operators etc. Counts all these from the program and then add one to that number. That will be the cyclomatic complexity of program.

So, from above all discussion, McCabe's cyclomatic complexity metric only considers complexity from control graph or conditional statements and not considers complexity is there interaction or coupling between two object classes.

Now, algorithm is purposed based on Field accesses through methods calls, Inheritance, Arguments, Return types, Exception, instruction type.

4.4 Proposed Algorithm for improving the concept of cyclomatic complexity

Purposed Algorithm for Coupling between object classes

Input: Any class file

Output: Total coupling in the class file

/**Eff is set for efferent coupling(set is java collection which holds only unique values), i is the index used, Interface contains the number of interfaces, Field contains the number of fields, Method contains the number of methods*, *filename is the name of input file, argumentstype is the type of argument, exceptiontype is the type of exception. Here class is being iterated using an API bcel.jar**/

1. Select a java class file and parse it using bcel API. //*bcel parses it and creates objects of class*
2. Set: filename = inputfilename;
3. Traverse the class.
4. foreach Interface ϵ class do
   registerCoupling (Interface);

   endfor

5. foreach Field ϵ class do
   registerCoupling (Field.returntype);

   endfor

6. foreach Method ϵ class *do //Identify different classes based on arguments, returntypes*
   registerCoupling (Method.returntype);

   registerCoupling (Method.argumentstype);

   registerCoupling (Method.exceptiontype);

```
            registerCoupling (Method.localargumentstype);
        foreach MethodInstruction
    /* if conditions can be determined by using api bcel.jar and calling its api*/
            if (instruction instanceof  LocalVariableInstruction)
              registerCoupling (LocalVariableInstruction type);
            endif
  if (instruction instanceof  ArrayInstruction)
      registerCoupling (ArrayInstruction type);
            endif
 if (instruction instanceof  FieldInstruction)
      registerCoupling (FieldInstruction type);
            endif
    if (instruction instanceof  InvokeInstruction)
        registerCoupling (InvokeInstruction type);
            endif
 if (instruction instanceof  INSTANCEOF)
    registerCoupling (INSTANCEOF type);
            endif
if (instruction instanceof  CHECKCAST)
   registerCoupling (CHECKCASTtype);
            endif
    endfor
endfor
```

7. Return Eff.size();

Methods definition:

```
registerCoupling (Type t){
      registerCoupling (className(t))
}
className(Type t){
          if (t is premetive)
                    return;
          endif
          if (t is array type)
                    return;
          endif
                    return name of type t ;
/*in java it can be t.tostring()*/
                              }
registerCoupling (String classname){
          if (classname is javaclass or classname is filename)
                    return;
           endif
          /*Add to set*/
          Eff.add (classname)
}
```

The algorithm here uses a library BCEL (byte code engineering library) which helps to understand and read the byte code of java file. It has API's which can read the class file line by line and gives the return type, no of interfaces etc. Here a class HashSet which will save all the unique external references to other classes is used. Since in java Hashset which is implementation of Set store only unique values. This data structure proved very helpful the intentions were to get unique references.

Also there is a method which has been frequently called throughout the algorithm is registerCoupling (). This method takes the class type or class name as input parameter and adds it to the set. This method has been overloaded, one taking a String parameter and other taking Type (BCEL API's class) which calls other method className(Type t) (reference: See methods section at the end of algorithm) which returns the name of the class and make sure that this is not a primitives type. Only non primitive's types are considered and in registerCoupling (String name) method, check whether it is a java API class or not if not then adds it to the set.

So, the algorithm starts with a java .class file (which is basically byte code) being passed as input and its name is stored in a variable. The purpose of this variable is, as CBO is number of unique external class references. So, this will not be taken in to consideration as a class name.

Since, CBO is total number of unique references, whether it is inheriting some other class or some local variable of other class type or exceptions or return type of a method or instance of other class algorithm tries to cover all the permutations.

In step 4 and 5 all the interfaces and all the class level fields respectively are considered and added their class to the set by calling method registerCoupling.

In step 6 it has started reading all the methods of the class. Firstly return type, exception type and argument types of this method are determined and these types are added to set again by calling registerCoupling method (). Now, method is traversed line by line and each line has various type of instructions. These can be anything like a local variable is declared etc. For these instructions BCEL has provided various classes such as LocalVariableInstruction (i.e. a local variable has been created) INSTANCEOF (i.e. whether a field or object is instance of some other class) and these classes are again added to set by calling registerCoupling method.

Finally, the length of this set is the final CBO of the input class.

Now, Final McCabe cyclomatic complexity number should be

Newcyclomatic complexity = Cyclomatic Complexity number + Coupling between object classes.

Now, consider a simple source's code example, which will more clearly explain the coupling concept.

Source code:

```
public class Example extends D {       // Inheritance
        public void callA(B b) {       // Argument
                A a = new A ();        // local variable
                try {
                        a.print(b);
      } catch (C c) {        //Exception
                System.out.println("Error");
  }
                        }
          public static void main(String aa[]) {
      Example e = new Example ();
                B b = new E ();     //local instruction
                        b.print ();
                        e.callA (b);
        }
                            }
```

```
class A {
public void print(B b) throws C {
System.out.println("In A");
}
}
class B {
public void print() {
System.out.println("In B");
}
}
class C extends Exception {
public void print() {
System.out.println("In C");
}
}
class D {
public void print() {
System.out.println("In D");
}
}
class E extends B {
public void print() {
System.out.println("In E");
```

```
        }
}
```

The above source code's consists of six classes having different-2 coupling factor based on various rules. Now, the focus is on Example.class file having McCabe cyclomatic complexity number is 2 due to try-catch block and coupling of this class file is 5 based on all the unique external class reference. The above source code's trying to include all the rules based on the definition of coupling between object classes as already discussed or explained in above algorithm like due to Argument passing, Inheritance, local instruction, local variable etc.

So, final new cyclomatic complexity of this source code's should be 7(5+2) instead of 2.

CHAPTER 5

SNAPSHOTS AND RESULTS

The purpose is to improve the cyclomatic complexity concept for object-oriented concepts. It is very necessary to measure the exact complexity of software so that accurate assumption can be made about quality attributes.

So, in project we want to firstly calculate McCabe cyclomatic complexity and then coupling between object classes based on number of external unique references. So, for that firstly browse the file or choose the file from the options for which want to calculate the cyclomatic complexity along with coupling between classes.

Snapshots for projects:

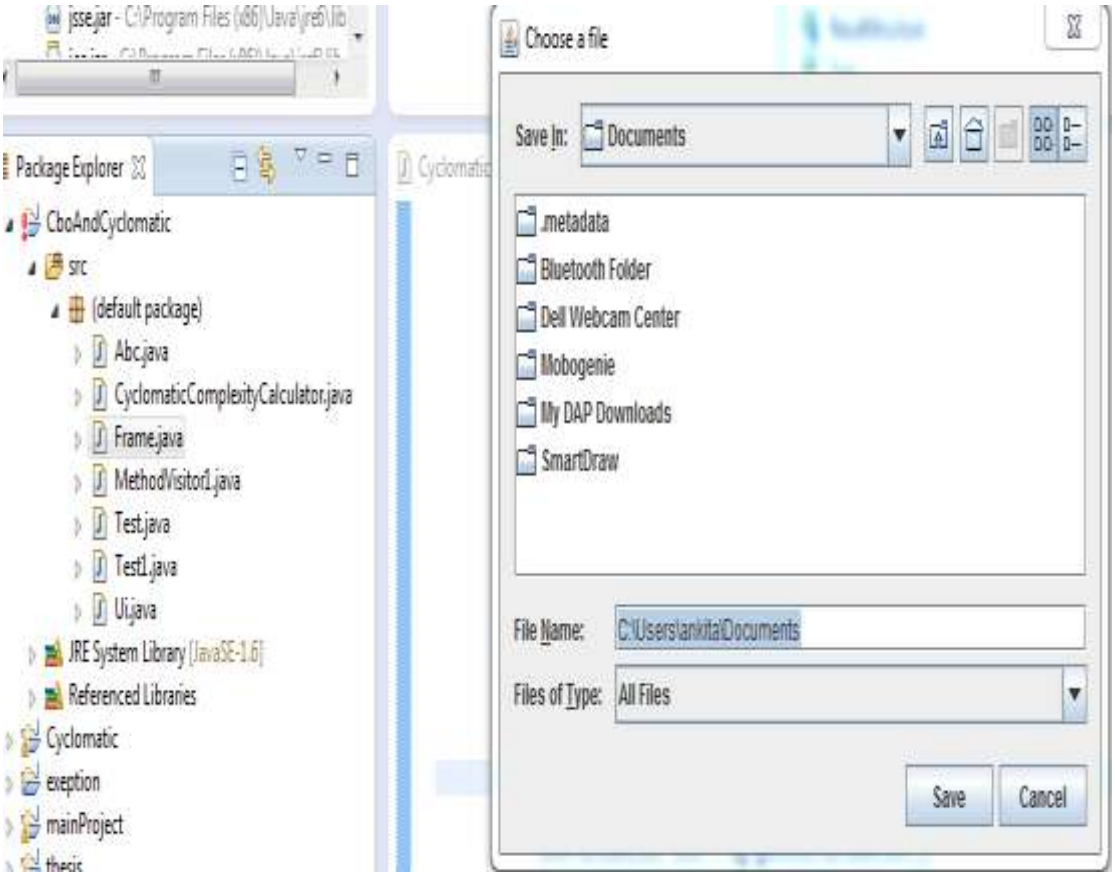


Fig.5.1 Browsing window

So, firstly browse the file or choose the file from the options. Firstly there is need to upload a project which consists of many java files using the first window. So, for that firstly open the eclipse and that window will appear after the running of that particular project.

That window is developed using java and hence the window appears after browsing has been developed using swing component of java. JFileChooser is the class which has been used to provide this basic graphical user interface (GUI). JFileChooser is an API to show the dialog box containing file chooser. It also provides various options like event handling option. For example if submit button is selected or cancel button is selected. We can write business logic on the basis of these decisions. So, with this option browse a file then select a file, after that it start reading that file using file system of java API.

Now, first snapshot shows the first window that will appear after running a project.

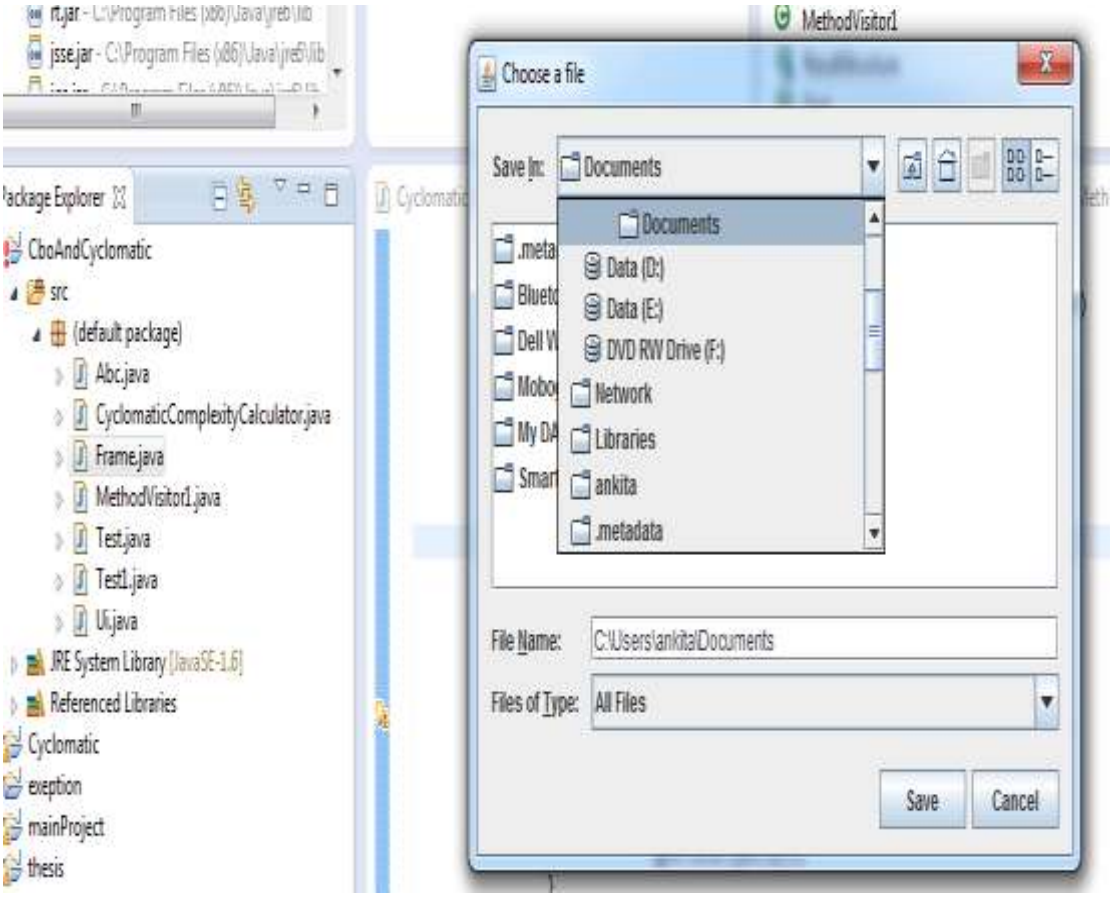


Fig.5.2 Second window

This window will appear when we want to select a particular project for operation. That project can be anywhere (like on desktop and at the other place in computer). Choose the location where currently project is placed.

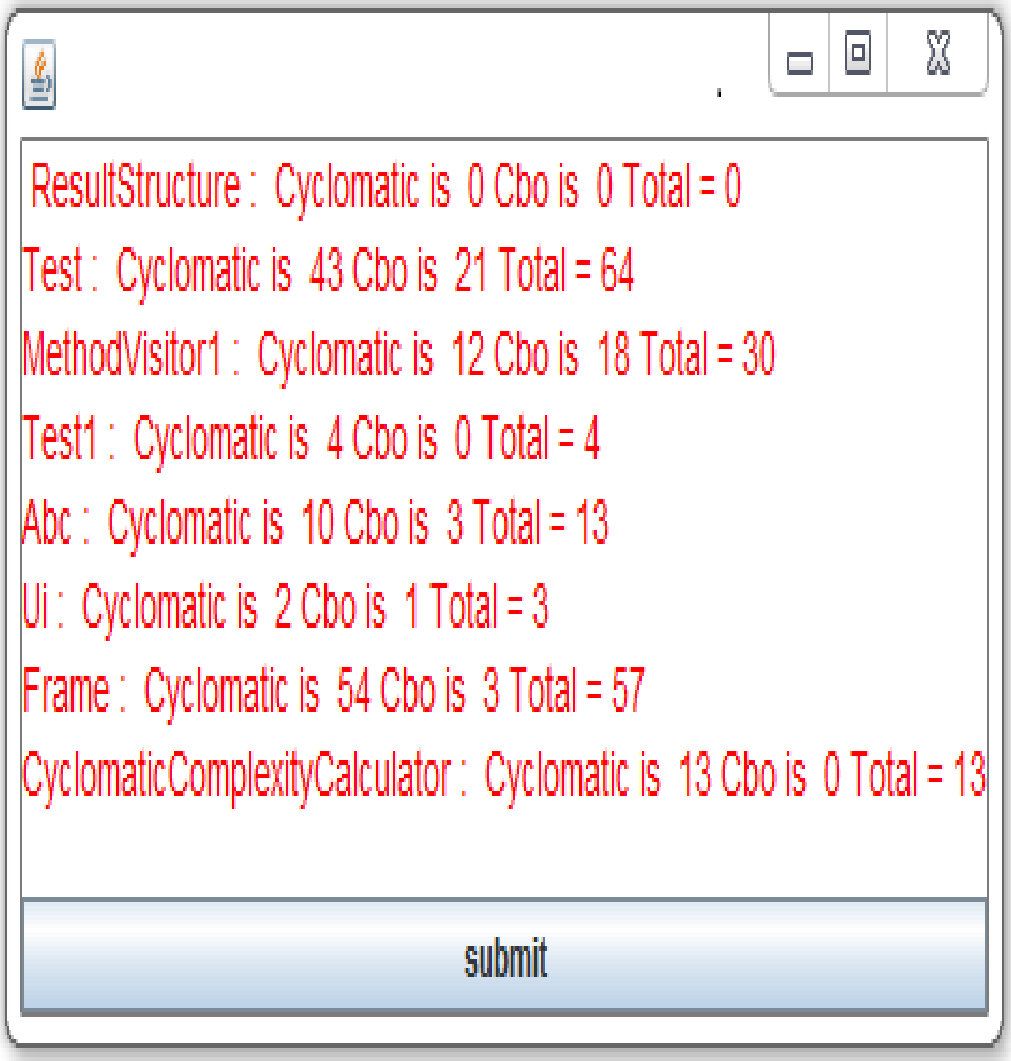


Fig.5.3 Output Window

Fig.5.3 shown above is the final window. The project consists of eight java files named ResultStructure, Test, methodVisitor1, Test1, Abc, Ui, Frame and CyclomaticComplexityCalculator. The purpose is to find the cyclomatic complexity along with coupling between all these classes. All these object classes are coupled to each other. So, firstly this window include the cyclomatic complexity number

corresponding to each java file then coupling between object classes and after that the total complexity of each file.

After calculating the above features if we again want to calculate these features for another file, and then click on submit button. Window will get opened and we can again browse the file or choose the file.

Table 5.1 New cyclomatic complexity of different API classes

S. No	Classes	Cyclomatic complexity number	Coupling between object classes	New Cyclomatic complexity number
1	org.apache.bcel.classfile.AccessFlags	4	0	4
2	org.apache.bcel.classfile.Attribute	18	21	39
3	org.apache.bcel.classfile.AttributeReader	1	2	3
4	org.apache.bcel.classfile.ClassFormatException	1	0	1
5	org.apache.bcel.classfile.ClassParser	17	6	23
6	org.apache.bcel.classfile.Code	20	7	27
7	org.apache.bcel.classfile.CodeException	4	5	9
8	org.apache.bcel.classfile.Constant	13	17	30
9	org.apache.bcel.classfile.ConstantClass	1	5	6
10	org.apache.bcel.classfile.ConstantCP	1	2	3
11	org.apache.bcel.classfile.ConstantDouble	1	4	5
12	org.apache.bcel.classfile.ConstantFieldref	1	2	3
13	org.apache.bcel.classfile.ConstantFloat	1	4	5
14	org.apache.bcel.classfile.ConstantInteger	1	4	5
15	org.apache.bcel.classfile.ConstantInterfaceMethodref	1	2	3
16	org.apache.bcel.classfile.ConstantDouble	1	4	5
17	org.apache.bcel.classfile.ConstantLong	1	4	5
18	org.apache.bcel.classfile.StackMapEntry	12	3	15

19	org.apache.bcel.classfile.ConstantClass	1	5	6
20	org.apache.bcel.classfile.LineNumber	2	2	4
21	org.apache.bcel.classfile.InnerClasses	6	4	10
22	org.apache.bcel.classfile.ClassParser	17	6	23

The table 5.1 shown above consists of various classes which have been taken from BCEL (Byte code engineering library) API. This API has been used in proposed algorithm. Firstly cyclomatic complexity number is calculated according to cyclomatic complexity concept (i.e. conditional statements) in first column. Along with a prediction can be made from this number about the quality of software, about maintenance cost, reliability, understanability and about other quality attributes etc. But it does not calculate the exact software complexity. There is a need to consider other factors to calculate the exact complexity of software so that an accurate prediction and assumption can be made from the exact complexity number. Cyclomatic number does not consider any complexity when a code interacts with other code e.g. Class A is calling some function of class B. For these interactions, there is a another factor i.e. CBO(complexity between objects).CBO is nothing but just the number of unique references i.e. a class has to other external classes by any means say it can inherit other class or some of its method has a return type of some external class etc . Other possibilities will be discussed in the algorithm for calculating CBO e.g. Class A inherits Class B and it has some function whose return type is Class C. In that case CBO for Class A will be 2.

So, in second column coupling between object classes is calculated based on all the unique external class references because cyclomatic complexity metric does not include the complexity if there is an interaction between object classes. Now, table includes various classes in which proposed algorithm has been applied and a new approach cyclomatic number plus coupling between classes has been applied for improving the concept of cyclomatic complexity.

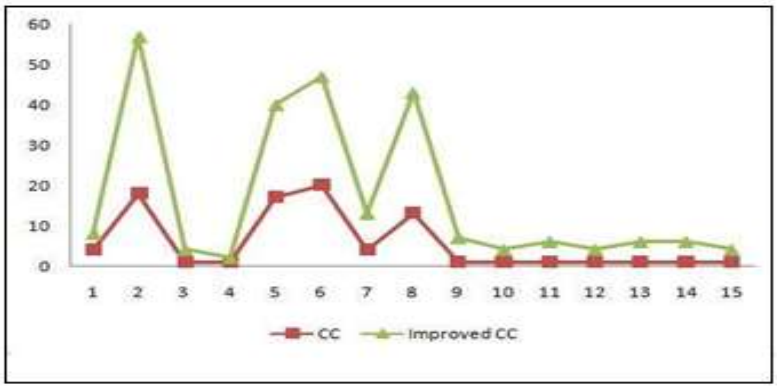


Fig.5.5 Improved graph for cyclomatic complexity.

The results discussed in the table 5.1 can also be depicted through a graph shown in the figure 5.5 where the graph shows the actual improved cyclomatic complexity of the various BCEL API classes taken.

CHAPTER 6

CONCLUSION AND FUTURE SCOPE

After a lot of survey about the software complexity metrics, Cyclomatic complexity measure of software complexity is strongest metric among LOC, Halstead's. It is language independent metric. Till now it is used to calculate the software complexity only through control flow graph (via conditional statements) but there is a need to include the concept of coupling with cyclomatic complexity concept. There was no concept of object oriented and structured languages at the time when Cyclomatic complexity was introduced. Because of ignorance of this concept in Cyclomatic complexity it was not considered as a best metric to measure the complexity of programming language like OOPs. So, it is very necessary to measure the complexity if there is an interaction between two object classes. So, for that one algorithm is purposed, how can calculate the coupling between object classes for object-oriented programming and then, used that concept in cyclomatic complexity for the improvement of cyclomatic complexity concept.

That algorithm is based on number of external unique class references and proposed on basis of other parameters like Field accesses, through methods calls, Inheritance, Arguments, Return types, Exception, instruction type. There are some benefits of improving the concept of cyclomatic complexity by proposing the algorithm of coupling between object classes and adding that coupling concept in cyclomatic complexity. The benefits are we are calculating the exact complexity of software. Software complexity is directly related to various quality attributes like maintenance cost, understanability, reusability, testing effort, time prediction and many others also.

That work is performed on many projects consisting of many classes. One example of API i.e. BCEL is already included in this thesis work and that work is included or shown in the form of graph and table.

The proposed approach is to firstly calculate the cyclomatic complexity according to its old concept i.e. via decisional statements, then calculate the interaction or coupling between object classes based on external unique references as already said. Now, add

this CBO in to cyclomatic complexity number that will be the final improved cyclomatic complexity.

In future, the respective approach can be stepped forward to consider the more accurate aspects of the coupling and improve the overall cyclomatic complexity of the software or system.

Second work can be the upper limit according to presented approach need to be revised in future as the previous approach ranges from 1 to 10 but this method leads to very high complexity in special cases.

REFERENCES

[1] T. B. Hilburn and D. J. Bagert, "A Software Engineering Curriculum Model", *29th lEEE Frontiers in Edticfition Confcrcnca*, 1999.

[2] M. Iqbal and M. Rizwan, "Application of 80/20 Rule in Software Engineering Waterfall Model", *IEEE International Conference on Information and Communication Technologies*, pp. 223-228, 2009.

[3] N. H. Madhavji, "The process cycle", *Software Engineering Journal*, pp. 234-242, 1991.

[4] C. Chug and M. Lee, "Inheritance-based Object-Oriented Software Metrics", *IEEE 10th International Conference on Computers, Communications and Automation,* pp.628-632, 1992.

[5] D.1. De and H. Perera, "Applicability of Three Complexity Metrics", *IEEE International Conference on Advances in ICT for Emerging Regions*, pp.82-88, 2012.

[6] U. Chhillar and S. Bhasin, "A new weighted composite complexity measure for object-oriented systems", *International journal of information and communication technology research*, pp. 101 -108, 2011.

[7] J. K. Kearney, "Software complexity measurement", *Communications of the ACM*, voI.29, pp. 1044-1050, 1986,

[8] *IEEE Standard Glossary of Software Engineering Terminology*, pp. 1-84, 1990.

[9] R. D. Banker, S. M. Datar and D. Zweig, "Software Complexity and Maintainability"[online].Available:file:///C:/Users/ankita/Downloads/79e41510fb40c9309e%20 (1).pdf.

[10] M. R. Woodward, M. A. Hennell and D. A. Hedley, "A measure of control flow complexity in program text," *IEEE Transactions on Software Engineering*, Vol. 5, No. 1, pp. 45–50, 1979.

[11] Metric Plug-in [online].Available: http://metrics.sourceforge.net/, 2014.

[12] A. Madi, "On the Improvement of Cyclomatic Complexity Metric**"** *International Journal of Software Engineering and Its Applications,* Vol. 7, No. 2, 2013.

[13] V. Y. Shen, T. Yu and S. M. Thebaut, "Identifying Error-Prone Software-An Empirical Study", *IEEE Transactions on Software Engineering*, Vol. SE-11, No. 4, pp.317 – 324,1985.

[14] S. Yu and S. Zhou, "A Survey on Metric of Software Complexity", *IEEE International Conference on Information Management and Engineering (ICIME),* Chengdu, pp. 352 – 356, 2010.

[15] B. Silva, C. SantAnna, C .Chavez and A .Garcia, "Concern-Based Cohesion: Unveiling a Hidden Dimension of Cohesion Measurement*", IEEE 20th International Conference on Program Comprehensive*, pp.103-112, 2012.

[16] G. Jay, J. E. Hale, R. K. Smith, D. Hale, N. A. Kraft and C. Ward, "Cyclomatic Complexity and Lines of Code: Empirical Evidence of a Stable Linear Relationship", *J. Software Engineering & Application*, pp.137-143, 2009.

[17] B. Curtis, S.B. Sheppard, P. Milliman, M. A. Borst, and T. Love," Measuring the Psychological Complexity of Software Maintenance Tasks with the Halstead and McCabe Metrics", *IEEE Transaction on Software Engineering*, pp.96-104, 1979.

[18] T. McCabe, "Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric," *NIST Special Publication*, pp. 500-235, 1996.

[19] M. S. Sarwar, I. Ahmad, S. Shahzad, "Cyclomatic Complexity for WCF: A Service Oriented Architecture", *IEEE Transactions on Software Engineering,* 2012.

[20] T. McCabe, "A Complexity Measure", *IEEE Transactions on Software Engineering*, pp.308-320, 1976.

[21] V. katiyar, "Implementation of Cyclomatic Complexity Matrix", *Journal of Nature Inspired Computing (JNIC)*, pp.26-29, 2013.

[22] M. Shepperd, "A critique of cyclomatic complexity as software metric", *Journal of Software Engineering*, pp.30-36, 1988.

[23] G. K. Gill and C. F. Kemerer, "Cyclomatic Complexity Density and Software Maintenance Productivity", *IEEE Transactions on Software Engineering,* Vol. 17, No. 12, pp.1284 – 1288, 1991.

[24] M. S. Sarwar, S. Shahzad and I. Ahmad, "Cyclomatic Complexity: The Nesting Problem", *8th International Conference on Digital Information Management,* pp.274-279, 2013.

[25] K. O. Emery and B. K. Mitchell, "Multi-level software based testing based on Cyclomatic Complexity", *IEEE National Conference on Aerospace and Electronics,* vol.2, pp.500 – 507, 1989.

[26] N. I. Enescu, D. Mancas, E. I. Manole, and S. Udristoiu, "Increasing Level of Correctness in Correlation with McCabe Complexity", *International Journal Of Computers*, pp.63-74, 2009.

[27] J. J. Vinju and M. W. Godfrey, "What Does Control Flow Really Look Like? Eyeballing the Cyclomatic Complexity Metric", *IEEE 12th International Working Conference on Source Code Analysis and Manipulation*, pp.154-163, 2012.

[28] MetricPlug-in[online].Available: http://sourceforge.net/projects/metrics2/files/stats/timeline, 2014.

[29] S. S. Rathore and A. Gupta, "Investigating Object-Oriented Design Metrics to Predict Fault-Proneness of Software Modules", pp. 1-10, 2012.

[30] A. Yadav and R. A. Khan, "Does Coupling Really Affect Complexity", *International conference on Computer and Communication Technology*, pp. 583-588, 2010.

[31] H. M. Kaung, N. S. Kham and N. L. Thein, "To Visualize the Coupling among Modules", *6th Asia –Pacific Symposium on Information and Telecommunication Technologies*, pp.111-116, 2005.

[32] B. Souley and B. Bata, "A Class Coupling Analyzer for Java Programs", *West African Journal of Industrial and Academic Research*, Vol.7, No. 1, 2013.

[33] N. S. A. A. Bakar and C.V. Boughton, "Validation of Measurement Tools to Extract Metrics from Open Source Projects", *IEEE Conference on Open Systems*, pp.1-6, 2012.

[34] V. Krishnapriya and K. Ramar," Exploring the Difference between Object Oriented Class Inheritance and Interfaces Using Coupling Measures", *International Conference on Advances in Computer Engineering*, pp.207-211, 2010.

[35] C. Ma, C. K. Chang, and J. Cleland-Huang, "Measuring the Intensity of Object Coupling in C++ Programs", *25th Annual International Conference on Computer Software and Applications*, pp.538-543, 2001.

LIST OF PUBLICATIONS

Published/Accepted

[1] Ankita Garg and Vinod K.Bhalla, "A Comparative Study of Software Complexity Metric", in Engineering Sciences International Journal, International Conference on Mathematics and Engineering Science, Vol.2, Chitkara University, Chandigarh, pp. 241-245, March 2014[Published].

[2] Ankita Garg and Vinod K.Bhalla, "Proposed Approach and Algorithm for Estimating the Cyclomatic Complexity in Object-Oriented Programming", IEEE International Conference on Computer, Communication and Control Technology, Langkawi, Kedah, Malaysia, 2-4 Sept 2014[Accepted].

Communicated

[1] Ankita Garg and Vinod K.Bhalla, "Framework for estimating Cyclomatic Complexity for Object-Oriented Programming", in International Conference on Soft Computing Techniques for Engineering and Technology (ICSCTET), Graphic Era Hill University, Nanital, 7-8 August 2014.