ABSTRACT

Measurement in software engineering remains an arduous activity. Various kinds of difficulties arise in the different phases of the software development life cycle. Measuring software complexity plays an important role to meet the demands of complex software. Cyclomatic complexity is one of the most used metric among the other metric like Line of code, Halstead’s measure and cyclomatic complexity. From the several software metrics available we are particularly interested in studying complexity metrics and find out how they can be used to evaluate the complexity of business processes.

Keywords---- Requirements Traceability, Metrics, LOC, Test case, Defects.

I. INTRODUCTION

Software engineering differs from other engineering disciplines in a number of aspects that have important consequences on software measurement. Software development is a highly iterative, dynamic process that requires constant feedback to gauge progress and make necessary adjustments to reach the required goal efficiently.

Software complexity is traditionally a direct indicator of software quality and cost. Logically, many of these measures have been shown to be correlated in some manner. Understanding these relationships is important to understanding and evaluating the metrics themselves and ultimately in reducing software development and maintenance efforts.[1]

II. LINES OF CODE (OR LOC)

This metric is used to measure the quantitative characteristics of program source code. LOC metric is based on counting the lines of the source code. The most common definition of LOC seems to count any line that is not a blank or comment line regardless of the number of statements per line. Logical LOC measures the number of statements, but their specific definitions are fixed to specific language for example, in C programming language logical LOC measure the terminating semicolon.[3].

III. FUNCTION POINT (FP)

FP metric was developed by Albrecht and he has proposed a measure of software size that can be determined early in the development process. FP approach is to compute the total function points (FP) value for the project, it depends on the counts of distinct (in terms of format or processing logic) types in the following five classes input/output, internal files external interface files, external inquiries.

FP = count total * [0.65 + 0.01 * (Fi)]

Here, the count total is the sum of all FP entries and the the Fi (i = 1 to 14) are complexity adjustment values.

IV. HALSTEAD METRICS

This metrics was introduced by Halstead in 1977 and this metrics is known as Halstead metrics.

He uses some primitive measures to develop expressions for program vocabulary (n), length (N), and volume (V) metrics. These are n1 (number of unique operators), n2 (number of unique operands), N1 (total frequency of operators), and N2 (total frequency of operands).[4]

Program Vocabulary (m) = n1 + n2
Program Length (N) = N1+N2
Program volume (V) = N log2 n
Program level (PL) = 1/ [(n1/2) *(N2/n2)]
Effort (e) = V/PL

V. MCCABE’S CYCLOMATIC COMPLEXITY

A cyclomatic complexity is a software metric which gives a quantitative measure of 4th logical complexity. When used in the context of the basis path testing method, it defines the number of independent paths [2].

The cyclomatic complexity of a section of source code is the count of the number of linearly independent paths through the source code. Thomas
McCabe introduced a metric in 1976 based on the control flow structure of a program or code. This metric is known as McCabe Cyclomatic complexity (CC) and it has been famous code complexity metric throughout since it was first introduced. Cyclomatic Number is one of the metric based on not program size but more on information/control flow. Cyclomatic complexity is defined as

\[ CC = E - N + P \]

Here,

- E = the number of edges of the graph
- N = the number of nodes of the graph
- P = the number of connected components

In case of connected graph

\[ CC = E - N + 2 \]

In simplified way it can be defined as

\[ CC = D + 1 \]

Here,

- D = the number of decision points in the graph

Here, some examples of how cyclomatic complexity is being calculated:

a) Ex. of CC=1:
Public Static Void Main()
{
    System.out.println("CC One ");
}

b) Ex. of CC=3:
Flow graph for this program will be as follows.

IF A = 99 THEN
IF B > C THEN
    A = B
ELSE
    A = C
ENDIF
ENDIF
PRINT A, B, C

In this example

\[ CC = \text{number of edges} - \text{number of nodes} + 2 \]
\[ CC = 8 - 7 + 2 \]
\[ CC = 3 \]

Now CC from simplified formula

\[ CC = \text{number of decision point} + 1 \]
\[ CC = 2 + 1 \]
\[ CC = 3 \]

\[ V(G) = 2 \text{ enclosed area} + 1 \]
\[ V(G) = 3 \]

VI. USES OF CYCLOMATIC COMPLEXITY

Cyclomatic Complexity can prove to be very helpful in developers and testers to determine independent path executions. Developers can assure that all the paths have been tested at least once. Also helps us to focus more on the uncovered paths and improve code coverage. Using this we can evaluate the risk associated with the application or program. These metrics are early in the software development life cycle reduces more risk of the program.

VII. TOOLS FOR CYCLOMATIC COMPLEXITY CALCULATION

There are many tools available for determining the complexity of the application and some complexity calculation tools are used for specific technologies.

- OCLint - Static code analyzer for C and Related Languages
- devMetrics - Analyzing metrics for C# projects. Reflector Add In - Code metrics for .NET assemblies
- GMetrics - Find metrics in Java related applications
- NDepends - Metrics in Java applications
VIII. CC’S IMPACT ON YOUR SOFTWARE QUALITY

The SEI provides the following basic risk assessment based on the value of code:

<table>
<thead>
<tr>
<th>Cyclomatic Complexity</th>
<th>Risk Evaluation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 to 10</td>
<td>Simple, low risk</td>
</tr>
<tr>
<td>11 to 20</td>
<td>Moderate complexity, medium risk</td>
</tr>
<tr>
<td>21 to 50</td>
<td>Complex, high risk</td>
</tr>
<tr>
<td>&gt; 50</td>
<td>Very high risk</td>
</tr>
</tbody>
</table>

IX. OBSERVATIONS AND ANALYSIS

In this section the focus is on the cyclomatic complexity. Why this is preferred over two metrics LOC and Halstead’s metric. Because all the three methods are used to measure the software complexity but only cyclomatic complexity metric is best among them because it considers the software complexity via number of decision statements or via control flow graph. And the Complexity metrics like Cyclomatic Complexity and Halstead metric which are used to predict the complexity of program. Still there is a problem along with cyclomatic complexity. McCabe’s cyclomatic complexity method is not used to calculate the software complexity if there is an interaction between two modules i.e. within a class or if there is an interaction between object classes means two object classes are communicating with each other and it is considered as if coupling or interaction is more than quality of software and other attributes like reliability, maintainability, modifiability get compromised. If there is some changes occur in one object class then simultaneous modification or changes occur in other class. This is a very important concept and must be in consideration.

Between LOC and Function Point metrics, Function Point metric is widely used to measure the size of code because it is language dependent and easily understood by non technical users. And the Complexity metrics like Cyclomatic Complexity and Halstead metric which are used to predict the complexity of program.

X. CONCLUSION

Software complexity metrics have a tendency to be used in judging the quality of software development and one of the vital parts of the SDLC. Complexity is always considered as an undesired property in software since it is a vital reason of diminishing software quality.

Why this is preferred over two metrics LOC and Halstead’s metric. Because, all three methods are used to measure the software complexity but only cyclomatic complexity metric is best among them. Cyclomatic complexity helps the developers and testers to determine independent path’s complexity for its execution. The number of decision point (rank) increases the complexity of the code increases then the probability of errors and time required for it also increases.

This paper provides the importance and understanding the issues and hypothesis of various software metrics and how they can be used to evaluate the complexity of business processes that could lead to reduce software development cost and improve testing efficiency and software quality by evaluating software complexity metrics.

REFERENCES