Complexity Measuring Tool

Version 1.0

DATE: 22-SEP-2019

GROUP ID : SPM_19_RSE_WE-3

GROUP NAME: Eventors 4.0

IT Number	Name
IT17050272	D.Manoj Kumar
IT17143950	G.M.A.S. Bastiansz
IT17068192	S. Thavananthan
IT17032766	K. Bavanraj

Table of Contents

1.0 Introduction

In present much attention has been directed toward reducing software cost. To this end, programmers have attempted to find relationships between the characteristics of programs & the difficulty of performing programming tasks. The objective has been to develop measures of program complexity that can be used for cost projection & program and programmer evaluation.

Despite the growing body of literature devoted to their development, analysis & testing, program complexity measures have yet to gain wide acceptance. Early claims for the validity of the metrics have not been supported, and considerable criticism has been directed at the methodology of the experiments that support the measures. Nonetheless, new complexity measures continue to appear, and new support for old measures is earnestly sought.

Complexity measures have been developed without any particular use in mind. Lacking a theory of programming behavior, programmers have sought meaningful applications through experimentation. The results of these explorations are difficult to interpret and provide only weak support for the use of complexity measures. Until more comprehensive evidence is available, software complexity measures should be used very cautiously.

Many IT companies mainly use java & C++ to create their programs. Regarding to reduce the cost of maintenance, the company needs to measure the complexity of Java & C++ programs they create. Therefore we, Eventors 4.0 creates this Complexity Measuring Tool which can easily measure the complexity of a program considering 4 main metrics namely,

1. Size
2. Type & the nesting level of controlling structures
3. Inheritance
4. Recursion

2.0 Overview

Using this Complexity Measuring Tool, programmer can measure a program considering 4 different metrics

2.1 Metrics Used

2.1.1 Measuring the Complexity of a Program Statement due to Size(Cs)

- A constant value of “2” is added to Cs at the detection of the following:
    1. Reference(&) and dereference(*)
    2. ‘new’, ‘delete’, ‘throw’, & ‘throws’ key words

- A constant value of “1” is added to Cs at the detection of the following:
    1. Arithmetic operators: {+ - * / % ++ -- }
    2. Relation operators: { == != > < >= <= }
    3. Logical operators: { && || ! }
    4. Bitwise operators: { | ^ ~ << >> >>> <<< }
    5. Miscellaneous operators: { , -> . ::}
    6. Assignment operators: { += -= *= /= = >>>= |= &= %= <<= >>= ^= }
    7. Key words: { void,double,int,float,string,printf,println,cout,cin,if,for,while,do-while,switch,case, etc}
    8. Manipulators: {endl , \n, etc}
    9. Text inside a pair of double quotes: {“ ”}
    10. Class, method, object, variable, & array names
    11. Numeric values

Note: Key words such as public, static, else, try and return are not considered under the size complexity factor.

The (*) sign used in the declaration of C++ pointer is not a dereference operator. It is just a similar notation that creates a pointer.

2.1.2 Measuring the Complexity of a Program Statement due to Type of Control Structure

- For a program statement with a **conditional control structure***(if)*, a weight of “1” is assigned for condition and for each **logical** (‘&&’ and ‘||’) or **bitwise** (‘&’ and ‘|’) operator that is used to combine two or more conditions
- For a program statement with an **iterative control structure** such as a **‘for’, ‘while’, or ‘do-while’** loop, a weight of two is assigned for the **‘for’, ‘while’, or ‘dowhile’** loop and for each **logical** (‘&&’ and ‘||’) or **bitwise** (‘&’ and ‘|’) operator that is used to combine two or more conditions.
- A weight of one is assigned for a program statement with a ‘**catch’** statement.

- A weight of n is assigned for a program statement with a ‘**switch’** statement with n number of cases
- A weight of **zero** is assigned for all the other program statements in a program.

2.1.3 Measuring the Complexity of a Program Statement doe to nesting of Control Structures(Cnc)

- A weight of zero is added for program statements which does not contain **any level of nesting**.
- A weight of one is added for program statements which are at the **outer most level of nesting**.
- A weight of two is added for program statements which are at the **next inner level of nesting**
- A weight allocated for the program statements is increased by one for **each level of nesting**

2.1.4 Measuring the Complexity of a Program Statement due to Inheritence(Ci)

- The complexity of all the program statements which belongs to a class is assigned the same weight that the class has due to its inheritance:

Complexity of a program statement of a class due to inheritance (Ci) = Complexity of the class due to its inheritance (CCi)

- Complexity of a class due to inheritance (CCi) is computed as follows:

Complexity of a class due to its inheritance (CCi) = Number of ancestor classes of the class + 1

2.1.5 Measuring the Complexity introduced due to Recursion(Cr)

- Double the Cps values derived for each program statement that belongs to a recursive method

2.2 User Access Level

- Everyone can use the application
- The uploaded program should not have unnecessary spaces

3.0 Getting Started

3.1 Installation & Login

3.2 System Menu