Code Complexity Measurement Tool

R.M.B. Devduni, Dasanayaka D.T.C. B, W.R.M.G.K. Wickramasinghe, E.M.M.P. Ekanayake
Faculty of Computing
Sri Lanka Institute of Information Technology
Bachelor of Science Special Honors Degree in Information Technology

***Abstract*— Software complexity is measuring characteristics including difficulty of implementing testing modifying or maintaining a program. The software engineering discipline has established some common measures of software complexity.**

Without the use of software complexity metrics, it can be difficult and time consuming to determine the where risk and cost emanates. When measuring complexity, it is important to look holistically at coupling, cohesion, SQL complexity, use of frameworks, and algorithmic complexity. It is also important to have an accurate, repeatable set of complexity metrics, Improve Code Quality, Reduce Maintenance Cost, Heighten Productivity and Increase Robustness

There are various complexity measuring tool,

1. Selenium

Selenium is an automated testing tool for web applications. It is an open source project that allows both testers and developers for developing functional tests in the browser. It allows developers to record the workflows, so that they can prevent future regressions of code. In addition, it works on any browser that supports JavaScript. Meet Architecture Standard.

2. SonarQube

SonarQube is an open-source platform used for continuous inspection of code quality to perform automatic reviews with static analysis of code to detect bugs, code smells, and security vulnerabilities.

3. Target process

Target process is a commercial Agile project management tool that allows following a Scrum, Kanban, or customized approach. It provides an intuitive and rich visual interface to manage your software development projects in a collaborative way.

Although there are several points that show the strength and validity of the proposed metric. However, there are also some drawbacks to the proposed measures, as follow:

- **The present method gives the complexity value in numerical terms, which are generally high for large programs. High complexity values are not desirable.**
- **Assigning the upper and lower boundaries for the complexity values.**
- **It is not possible to identify the underlying source of complexity with the proposed measure since it depends on several factors, such as; the number of methods, their internal architectures and the number of message calls.**

We try to develop our system through the previously introduced tools. We got the idea about how software complexity tool work.

We recommend the following in order to get over the draws back that were discussed above.

- **Extending the proposed metric to calculate dynamic complexity of an OO code.**
- **Developing a software tool to calculate our metric automatically.**
- **Investigating the upper and lower boundaries of the complexity values.**
- **Data from the industry should be applied to further empirical evaluation.**
- **The proposed metric should be studied in the light of making improvements to the remaining features of OO programs**

We develop a web-application code complexity measuring tool. We have 4 main functions.

- **Measuring the complexity of a program statement due to size, variables and methods**
- **Measuring the complexity of a program statement due to inheritance**
- **Measuring the complexity of a program statement due to coupling**
- **Measuring the complexity of a program statement due to control structures**
- **Measuring total complexity of a program statement and complexity of a program.**

Our system support file uploading including zip file and multiple file. An uploaded file into the system, we expected to measure the complexity for the following factor size, variables, methods, inheritance, coupling, and control structures.

IT solutions often use JAVA language and C++ to style their programs. They require to hunt out the complexity of like Size, Variables, Methods, Inheritance, Coupling, and Control Structures. This article contains descriptions of the complete system analysis, how we operate, and the present system we get ideas from. The methodology of our tool and the interface of our complex measurement tool. We explained how we managed the problems we faced and therefore the thanks to calculating the complexity of programming codes.

I. INTRODUCTION

In the fast-moving technological world, developers are one of the most important service providers in the field. And generally, they are seen developing codes. But at present developers can be seen spending a large amount of time in maintaining codes, which is more than the time that is being taken for developing

codes. Although it is such, developers rarely stop by and try to figure out where the problems are coming from that makes them take a long time in maintaining the codes. (Cho, Jeon, & Jeong, 2003, September.) Developers rarely or never try to identify that the problem is nothing other than software complexity. Developers do not stop and think that software complexity makes the job harder. That is the reason why learning about software complexity is important because it helps the developers increase the quality of the code that might make code maintenance easier. Software measurement is the fundamental aspect of any process or product for its success. Metrics are the unit of measurement which is used to describe the product, process and people of software engineering.
Our system support file uploading including zip file and multiple file. If there is zip file, we must unzip our file. After that an uploaded file into the system, we expected to measure the complexity for the following factor size, variables, methods, inheritance, coupling, and control structures and measuring total complexity of a program statement and complexity of a program.
Although there are several points that show the strength and validity of the proposed metric. (Lumley, 2011)However, there are also some drawbacks to the proposed measures, as follow:

- The present method gives the complexity value in numerical terms, which are generally high for large programs. High complexity values are not desirable.
- Assigning the upper and lower boundaries for the complexity values.
- It is not possible to identify the underlying source of complexity with the proposed measure since it depends on several factors, such as; the number of methods, their internal architectures and the number of message calls.

We try to develop our system through the previously introduced tools. We got the idea about how software complexity tool work.

We recommend the following in order to get over the draws back that were discussed above.
- Extending the proposed metric to calculate dynamic complexity of an OO code.
- Developing a software tool to calculate our metric automatically.
- Investigating the upper and lower boundaries of the complexity values.
- Data from the industry should be applied to further empirical evaluation.
- The proposed metric should be studied in the light of making improvements to the remaining features of OO programs.

II. Related Work/Literature Review

A. *Program complexity*

One of the most widely referenced sets of program complexity has been proposed by Harmeet and Gurvinder (Harmeet Kaur, 2016). Chris and Kemerer presented a paper (S. Chidambar & C. Kemerer, 1991) outlining six ways for use with object-oriented programming languages. Rajiv and Srikanth presented a paper (Inder M. Soi, 1985) about software complexity and software maintainability in commercial software environments. The metrics used in this study are given below.

B. *complexity of a program statement due to size and weight related to the size*

Complexity due to size we can calculate keyword, identifier, operators, numerical values and String literal of each line of code. Weight is total of keyword, identifier, operators, numerical values and String literal of program. In additional (Alexander, 1999) proposed to get complexity of size that were used in their study including.

C. *complexity of a program statement due to variables and weights related to the variable*

Variables of program (Tang MH & Kau MH, 1999) we can calculate all the composite and primitive data types of each code line. Weight of variables we can calculate total of all the variables in program. (Roger S. Pressman, 2001) proposed to calculate variables of program. It covers huge scope of complexity measuring.

D. *complexity of a program statement due to methods and weight related to the methods*

Calculating all the primitive data type parameters and composite data type parameters we can get complexity of methods. Weight of methods defines as the sum of complexities of all methods of a program. Studying paper (Li.W, 1998) we can get more knowledge about measuring methods of program.

E. *complexity of a program statement due to coupling and weight related to the coupling*

The hardest part is measuring the coupling in program. Coupling metric measures the number of method calls defines in methods of a class to methods in other classes, and therefore the dependency of local methods to methods implemented by other classes (Chengying, 2010). Weight of coupling can calculate by total of the coupling in program.

F. *complexity of a program statement due to control structures and weight related to the control structures*

Calculating all conditional keywords, Iterative keywords, and switch case keywords of program we can get complexity of control structures of program. (Niggl & Karl, 2005)

G. complexity of a program statement due to inheritance and weight related to the inheritance

Morris states that "inheritance hierarchies are optimized via a process called factoring. The purpose of factoring is to minimize the number of locations within an inheritance hierarchy in which a method is implemented. (K. Morris, 1989)

III. PROPOSED SYSTEM

We develop a desktop-based code complexity measuring tool. Accordingly, we develop measure the complexity due to following factors.

- Size, variable, method
- Inheritance
- Coupling
- Control Structure

A. Measuring complexity of Programme statement due to size, variable, method

The user can measure his submitted code according to the size component through the desktop-based code measuring tool which the complexity is focused on Java**.**

Only the lines which consist of declared or defined variables are considered under variable factor. Scope, primitive data type and composite data type variables are considered when computing**.** Based on the scope, variables are divided into 2 as global variables and local variables to which a weight of 2 and 1 are allocated for global variable and local variable respectively**.**

Only the lines which include method signatures are considered under the methods factor. In here, methods are divided into 2 categories as primitive data types and composite data types, where a weight of 1 is assigned for primitive data type and a weight of 2 is assigned for each method in a composite data type.

This section presents a desktop-based tool for size, variable, method measurement developed by our group members. This tool is static java source code analyzing java code (Wilkie & TJ, 2002 Oct 3). The tool accepts a zip/ .java source file as input. The desktop-based tool for size, variable, method is designed and develop with java. the architecture diagram of the tool is dedicated in figure1. The tool accepts input filled application files shown in the architecture diagram as a mentioned earlier the tool is restricted to accepted only .zip/.java source file as input.
The zip file extractor unzips the input files and retrieves java files from the folder. After that one by one the files are taken into buffer reader.
The user can browser required project location to select input java files.

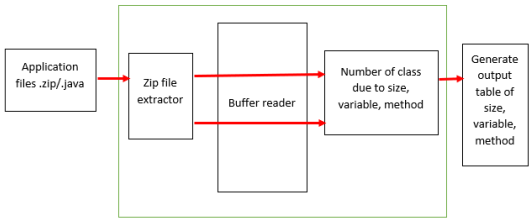

Figure 1 Architecture of size, variable, method

Complexity of a program statement due to its size, variable, method can be computed as follows,

- **Size**

Cs = (Wkw * Nkw) + (Wid * Nid) + (Wop * Nop) + (Wnv * Nnv) + (Wsl * Nsl)

CS = Complexity of a program statement due to its size
Wkw = Weight due to keywords or reserved words
Nkw = Number of keywords or reserved words in the program statement
Wid = Weight due to identifiers
Nid = Number of identifiers in the program statement
Wop = Weight due to operators
Nop = Number of operators in the program statement
Wnv = Weight due to numerical values or numbers
Nnv = Number of numerical values in the program statement
Wsl = Weight due to string literals
Nsl = Number of string literals in the program statement

- **Variable**

Cv = Wvs [(Wpdtv * Npdtv) + (Wcdtv * Ncdtv)]

Cv = Complexity of a program statement due to its variables
Wvs = Weight due to variable scope
Wpdtv = Weight of primitive data type variables
Npdtv = Number of primitive data type variables
Wcdtv = Weight of composite data type variables
Ncdtv = Number of composite data type variables

- **Method**

Cm = Wmrt + (Wpdtp * Npdtp) + (Wcdtp * Ncdtp)

Cm = Complexity of a line which includes a method signature
Wmrt = Weight due to method return type
Wpdtp = Weight of primitive data type parameters
Npdtp = Number of primitive data type parameters
Wcdtp = Weight of composite data type parameters
Ncdtp = Number of composite data type parameters

B. Measuring the complexity of a program statement due to inheritance

The systems based on object-oriented programming there is an important feature called Inheritance. Through this function, it calculates the complexity at method level regarding internal structure of methods and complexity of class hierarchies. In

addition to that through this function it indicates and supports the concept of "reusability".

This section present desktop-based tool for Inheritance measurement. The desktop-based tool for Inheritance is designed and develop with java. The architecture diagram of Inheritance measurement is dedicated in figure 2. The tool accepts input field show in the architecture diagram. The tool is restricted to accepted only .zip/.java files as input.

The zip file extractor unzips the input files and retrieves java files from the folder. After that one by one the files are taken into buffer reader.

The user can browser required project location to select input java files.

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

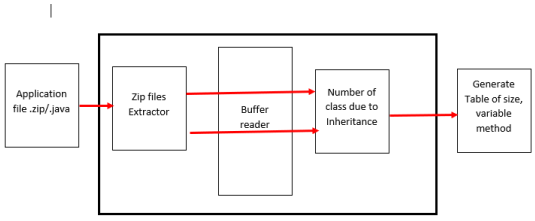

Figure 2 Architecture of Inheritance factor

C. Measuring the complexity of a program statement due to control structure

A control structure is a block of programming that analyzes variables and takes a direction in which to go based on given parameters. There are three basic control structures in a procedural language.

This section presents desktop-based tool for Control structure. The desktop-based tool for control structure is designed and develop with java. The architecture diagram of control structure measurement is dedicated by figure 3. The tool accepts input field show in the architecture diagram the tool is restricted to accept only. .zip/.java source files as Input.
For a program statement with a conditional control structure such as an 'if' condition, a weight of one is assigned for the 'if' 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 'do while' loop and for each logical ('&&' and '||') or bitwise ('&' and '|') operator that is used to combine two or more conditions. (Choi, 2003)
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.

In this tool the zip file extractor unzips the input files and retrieves java files from the folder. after that one by one the files are taken into the buffer reader. The user can browse required project location to select input .java files.

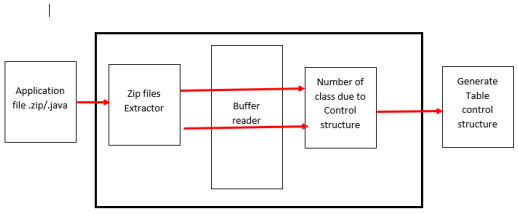

Figure 3 Architecture of Control structure factor

D. Measuring the complexity of a program statement due to coupling

Software complexity can be estimated using various factors of software. Coupling is one of the factors which affect quality of software.

This section present desktop-based tool for coupling. (Dao, Kermanshachi, Shane, & Anderson, 2016.)The desktop-based tool for coupling is designed and develop with java. The architecture diagram of coupling is measurement in figure 4. The tool accepts input field show in the architecture diagram the tool is restricted to accept only .zip/.java files source files as Input.

Complexity of program statement due to coupling (Ccp) is computed as follows.

Ccp = (Wr * Nr) + (Wmcms * Nmcms) + (Wmcmd * Nmcmd) + (Wmcrms * Nmcrms) + (Wmcrmd * Nmcrmd) + (Wrmcrms * Nrmcrms) + (Wrmcrmd *Nrmcrmd) + (Wrmcms

* Nrmcms) + (Wrmcmd * Nrmcmd) + (Wmrgvs *Nmrgvs) + (Wmrgvd * Nmrgvd) + (Wrmrgvs * Nrmrgvs) + (Wrmrgvd * Nrmrgvd)

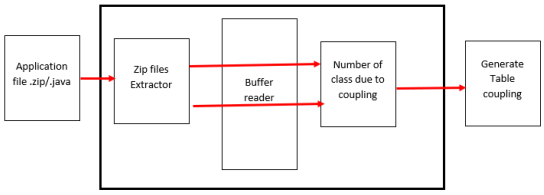

Figure 4 Architecture of Coupling

E. Interface of Code Complexity Measurement Tool

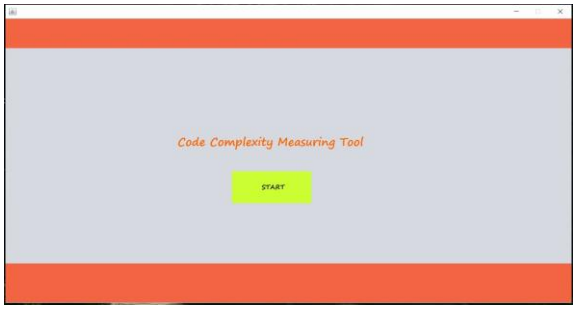

Figure E.1: Home Page of code complexity measurement tool

Figure E.2: All factors of code complexity measurement tool

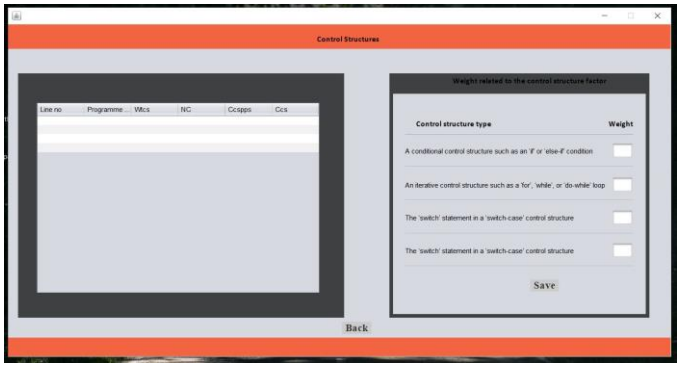

Figure E.3:control structure of code complexity measurement tool

Figure E.4: coupling of code complexity measurement tool

Figure E.5 All factors of coupling code complexity measurement tool

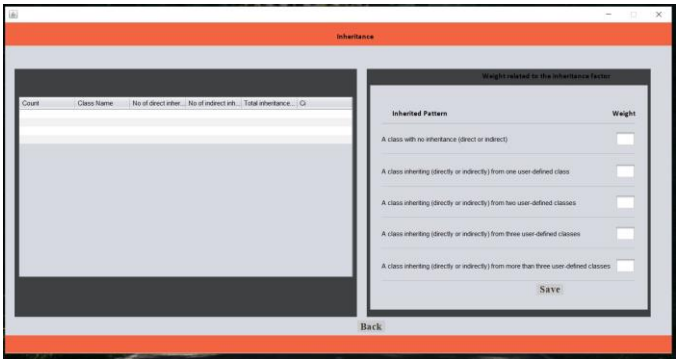

Figure E.6: Inheritance of code complexity measurement tool

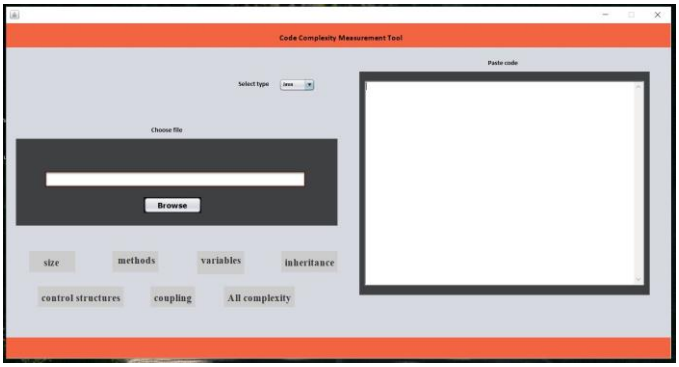

Figure E.7: Main interface od code complexity measurement tool

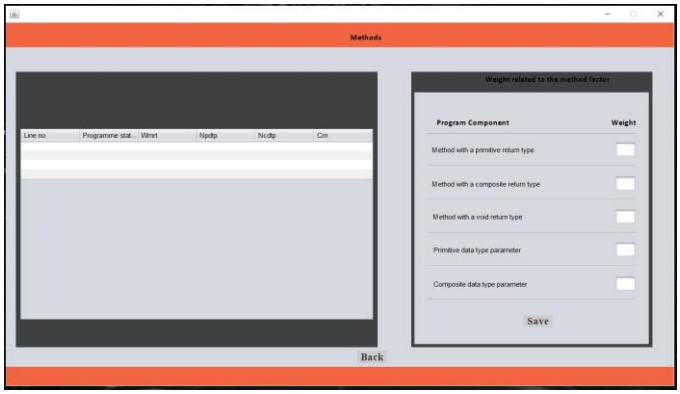

Figure E.8: methods factor of code complexity measurement tool

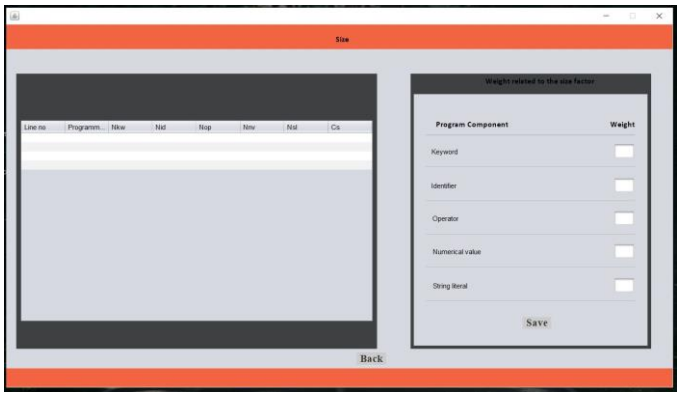

Figure E.9: size factor of code complexity measurement tool

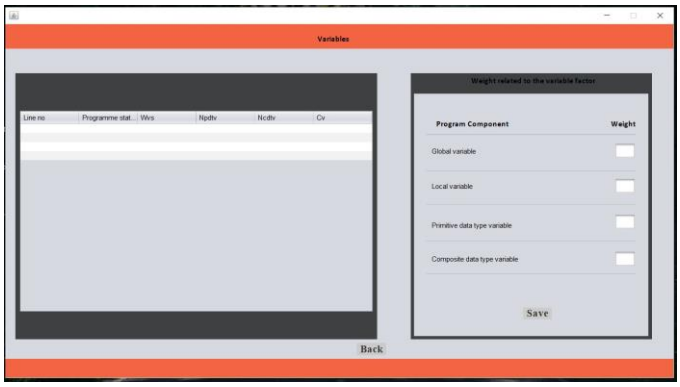

Figure E.10: variable factor of code complexity measurement tool

IV. METHODOLOGY

Methodology section describes each functionality in the code complexity measuring tool. Particularly the process of detecting each graphical (Dao, Kermanshachi, Shane, & Anderson, 2016) content in a document, analyzing, reading the contents and the technologies and tools used in implementation.

First to create this system, we have used different technologies and tools. We used Eclipse IDE as our developing tool, phpMyAdmin and selenium as a testing tool.

- NetBeans

During the project in order to build our system we used NetBeans IDE. Using this tool, we can get benefits such as profiling and debugging tools, dynamic language support, easy to create interfaces etc.

- Selenium

During the implementation of this project, this open source platform was used for to analyses and measure the source code quality and detect bugs.

- GitHub

We used GitHub for as a project management tool and a tool for to integrate and collaborate with the other group members. This is an ideal tool for agile projects.

•phpMyAdmin

To this project we need a database to store some files. In order to do that we chose phpMyAdmin as our database. So that we can handle easily.

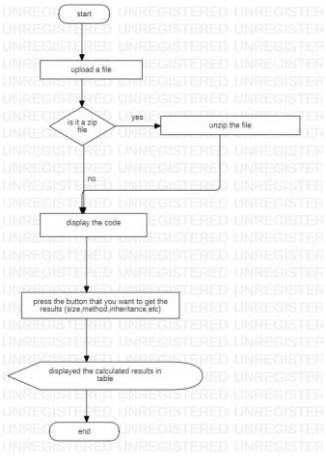

Figure 4.1: Flow chart for complexity tool

According to this system Fig 5.1. flow chart shows the process of the complexity measuring system and working process. After running the project user has to press start button to continue the system.

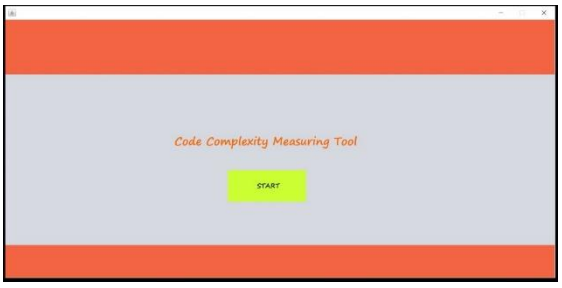

Figure 4 .2: .2: start the complexity measuring tool

This code complexity measuring tool contains the main process of the first screen which uploads the file that contains the Java code. In our system tool is restricted to accepted only .zip/.java source file as input.

If your file is a zip file, then the zip file extractor unzips the input files and retrieves the java file from the folder then the file will save in your computer. After that one by one files are taken into buffer reader. After that the user can browse the required project location and select input java file.

Figure 4.3: browse the file

By pressing open button, the code displayed in the screen.

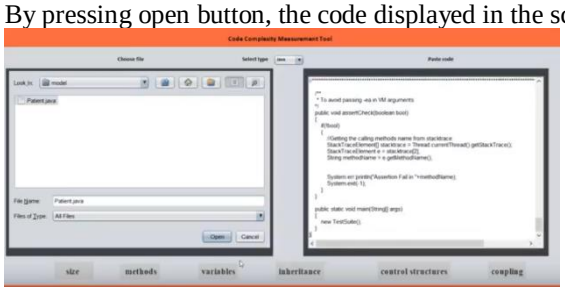

Figure 4.4: display the code in screen

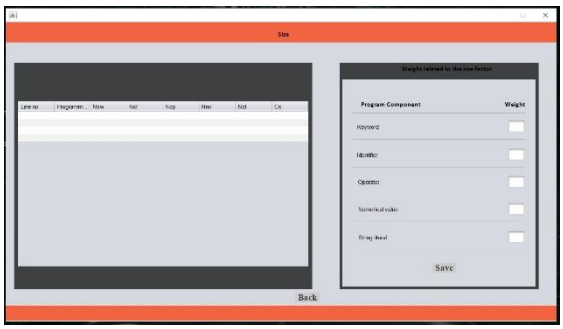

Figure 4.5 weight related to the size factor

V. Result Of Discussion

In this section, we discussed about all the results obtained from the implemented components of the system. The goal of this research is to develop a system to measure the code complexity due to size, variables, methods, inheritance, control structures and coupling.

When uploading a java file, you need to upload it as a single file and the developed system must have a zip or unzip file. As a result of this system calculate the derivation of complexity due to size, variable, methods, inheritance, coupling and control structures.

A. Measuring the complexity due to size

Cs = (Wkw * Nkw) + (Wid * Nid) + (Wop * Nop) + (Wnv * Nnv) + (Wsl * Nsl)

Using this formula user can get the results of complexity due to size and weight related to the size can be identify by sum like keywords, identifiers, operators, numerical values and string literal.

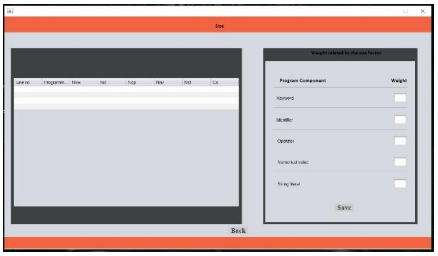

Figure 5.A: shows the results of complexity due to size and weight related to the size.

*B. **Measuring the complexity due to variables***

Cv = Wvs [(Wpdtv * Npdtv) + (Wcdtv * Ncdtv)]

Using above formula user can calculate the complexity of variables. Weight related to the variable factor can be find as a global and local variable, primitive data type variable and composite data type variable. The results of this variable factor can be displayed like,

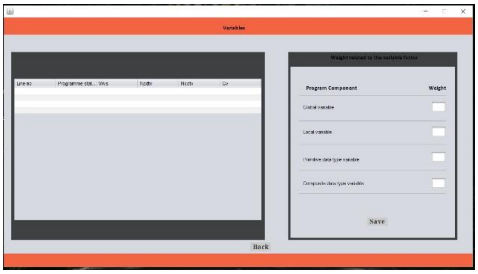

Figure 5.B: results of complexity due to variable and weight related to variable.

*C. **Measuring the complexity due to methods***

Cm = Wmrt + (Wpdtp * Npdtp) + (Wcdtp * Ncdtp)
Using above formula can get the results of the complexity due to methods. The weight of related method can be identified as Method with a primitive return type, Method with a composite return type, Method with a void return type, Primitive data type parameter, Composite data type parameter.

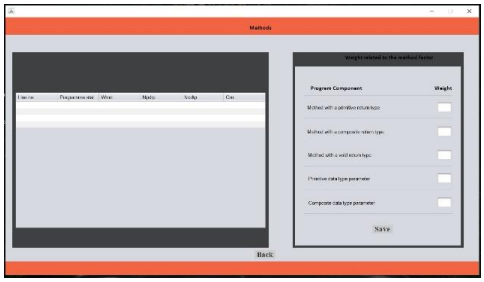

Figure 5.C: results of complexity due to methods and weight related to methods

D. Measuring the complexity due to inheritance

Ti = Ndi + Nidi
Ti = Ci

Using following formula user can get the result of inheritance, sum of
No of direct and indirect inheritance. The results can be show as following table.

ID	Test Input	Expected output	Actual output	results
1	Upload java file for variable	Show the total value of complexity due to variable	Actual output as expected	pass
2	Upload java file for inheritance	Show the total value of complexity due to inheritance	Actual output as expected	pass
3	Upload java file for control structures	Show the total value of complexity due to control structure	Actual output as expected	pass
4	Upload java file for coupling	Show the total value of complexity due to coupling	Actual output as expected	pass

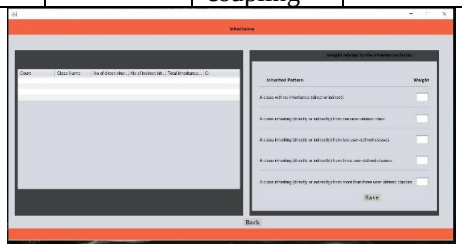

Figure 5.D: results of complexity due to inheritance

E. Measuring the complexity due to control structures

Ccs = (Wtcs * NC) + Ccspps

Can be obtained using above formula to calculate the complexity of Control Structure. Weight is the main flexibility feature of the system. The default values are already defined by the developed system. The user can enter any number, but you cannot enter the negative values and save. After that, the results returned will be updated according to the given values. The results of the complexity due to control structure and the total complexity will be updated in the table.

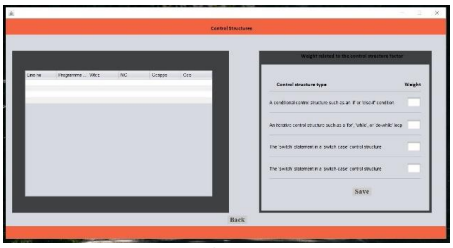

Figure 5.D: results of complexity due to control structure

F. Measuring the complexity due to coupling

Ccp = (Wr * Nr) + (Wmcms * Nmcms) + (Wmcmd * Nmcmd) + (Wmcrms * Nmcrms) + (Wmcrmd *Nmcrmd)+(Wrmcrms * Nrmcrms) +(Wrmcrmd*Nrmcrmd)+(Wrmcms * Nrmcms) + (Wrmcmd * Nrmcmd) + (Wmrgvs *Nmrgvs) + (Wmrgvd * Nmrgvd) + (Wrmrgvs * Nrmrgvs) + (Wrmrgvd * Nrmrgvd)

Using above formula user can obtain the complexity due to coupling and Change the weight values of the coupling factor for the calculations. When the user can add any numeric value for this weight, discard negative numbers. After adjusting the weight for the coupling.

Figure 5.F: results of complexity due to coupling

VI. CONCLUSION

This paper presents a desktop-based tool for measuring size, variable, method, control structure, Inheritance and coupling. The tool helps to software developer to maintain size, variable, method, control structure, Inheritance coupling values in the

standard range. The tool accepts java project as an input and computers all the values as mentioned in earlier. The tool is useful for offline computation and we can print results.

In future tool can be extended to work for other object-oriented programming language like .Net, c++, etc.

VII. REFERENCES

I. Banker, R.D., Datar, S.M., Kemerer, C.F. and Zweig, D., 1993. Software complexity and maintenance costs. *Communications of the ACM*, *36*(11), pp.81-95.

II. Healy, S.D. and Rowe, C., 2007. A critique of comparative studies of brain size. *Proceedings of the Royal Society B: Biological Sciences*, *274*(1609), pp.453-464.

III. Wilkie, F.G. and Harmer, T.J., 2002, October. Tool support for measuring complexity in heterogeneous object-oriented software. In *International Conference on Software Maintenance, 2002. Proceedings.* (pp. 152-161). IEEE.

IV. Choi, W.I., Jeon, B. and Jeong, J., 2003, September. Fast motion estimation with modified diamond search for variable motion block sizes. In *Proceedings 2003 International Conference on Image Processing (Cat. No. 03CH37429)* (Vol. 2, pp. II-371). IEEE.

V. Thomaz, S.M., Dibble, E.D., Evangelista, L.R., Higuti, J. and Bini, L.M., 2008. Influence of aquatic macrophyte habitat complexity on invertebrate abundance and richness in tropical lagoons. *Freshwater biology*, *53*(2), pp.358-367.

VI. Buys, L., Mengersen, K., Johnson, S., van Buuren, N. and Chauvin, A., 2014. Creating a Sustainability Scorecard as a predictive tool for measuring the complex social, economic and environmental impacts of industries, a case study:

VII. Ye, J., 1998. On measuring and correcting the effects of data mining and model selection. *Journal of the American Statistical Association*, *93*(441), pp.120-131.

VIII. Dao, B., Kermanshachi, S., Shane, J. and Anderson, S., 2016. Project complexity assessment and management tool. *Elsevier Journal of Procedia Engineering*, *145*, pp.491-496.

IX. Lumley, T., 2011. Complex surveys: a guide to analysis using R (Vol. 565). John Wiley & Sons.

X. Saponara, S., Blanch, C., Denolf, K. and Bormans, J., 2003, April. The JVT advanced video coding standard: complexity and performance analysis on a tool-by-tool basis. In IEEE Packet Video (pp. 98-109).

XI. Saponara, S., Blanch, C., Denolf, K. and Bormans, J., 2003, April. The JVT advanced video coding standard: complexity and performance analysis on a tool-by-tool basis. In IEEE Packet Video (pp. 98-109).

XII. Eriksson, J., Ollila, E. and Koivunen, V., 2010. Essential statistics and tools for complex random variables. IEEE Transactions on signal processing, 58(10), pp.5400-5408.

XIII. Kearney, J.P., Sedlmeyer, R.L., Thompson, W.B., Gray, M.A. and Adler, M.A., 1986. Software complexity measurement. Communications of the ACM, 29(11), pp.1044-1050.

XIV. McCabe, T.J., 1976. A complexity measure. IEEE Transactions on software Engineering, (4), pp.308-320.

XV. Lewis, F.I., Brülisauer, F. and Gunn, G.J., 2011. Structure discovery in Bayesian networks: An analytical tool for analysing complex animal health data. Preventive veterinary medicine, 100(2), pp.109-115.

XVI. Chen, Q., Ding, J., Cai, J. and Zhao, J., 2012. Rapid measurement of total acid content (TAC) in vinegar using near infrared spectroscopy based on efficient variables selection algorithm and nonlinear regression tools. Food chemistry, 135(2), pp.590-595.

XVII. Harmeet Kaur "Software Complexity Measurement: A Critical Review" International Journal of Engineering and Applied Computer Science 2016.

XVIII. S. Chidamber, and C. Kemerer, "Towards a Metrics Suite for Object Oriented Design,'' Object Oriented Programming Systems, Languages and Applications (OOPSLA), Vol 10, 1991, pp 197-211.

XIX. Inder M. Soi "Software complexity: An aid to software maintainability" Microelectronics Reliability 1985.

XX. Alexander Romanovsky "Class diversity support in object-oriented languages" Journal of Systems and Software 1999.

XXI. Tang MH, Kao MH. "An empirical study on object-oriented metrics". Proceedings 23rd Annual International Computer Software and Application Conference. IEEE Computer Society, 242-249,1999.

XXII. Roger S. Pressman: Software Engineering, A practioner's Approach, Fifth Edition,2001.

XXIII. Li. W. "Another Metric suit for object-oriented programming". The journal of system and software 44(2),155-162,1998.

XXIV. Chengying Mao "Control Flow Complexity Metrics for Petri Net-based Web service Composition" Journal of Software 2010.

XXV. Niggl, Karl-Heinz. (2005). Control structures in programs and computational complexity. Annals of Pure and Applied Logic. 133. 247-273. 10.1016/j.apal.2004.10.011.

XXVI. K. Morris, "Metrics for Object-oriented Software Development Environments," Master's Thesis, MIT, 1989.