VOL. 10, NO. 5, MARCH 2015 ISSN 1819-6608
ARPN Journal of Engineering and Applied Sciences


www.arpnjournals.com

EVALUATING METRICS AT CLASS AND METHOD LEVEL FOR JAVA PROGRAMS USING KNOWLEDGE BASED SYSTEMS

Umamaheswari E.[1], N. Bhalaji[2] and D. K. Ghosh[3]
[1]SCSE, VIT Chennai Campus, Chennai, India
[2]SSN College of Engineering, Kalavakkam, India
[3]V.S.B Engineering College, Karur, India

ABSTRACT

Software metrics is considered to be the most important tool in software process management. It is also measure of property for a specific piece of software [1]. Metrics also serves as a resource to anticipate and avoid the problems. Since there are only few measurement tools available, the need for metrics tool in testing the software is increasing. Although many metrics have been proposed by researchers they are used in isolation or ignored because they are not focused much. Therefore, an open source tool called "JAM (java metrics)" is to be developed to calculate various metrics and to display the metrics in graphical representation for java code. This learning tool allows software engineers to measure their code and to improve their software quality. It calculates the metrics at class level and method level. This tool also provides some basic information about the metrics calculated.

Keywords: software metrics tool, JAVA, class level metrics.

1. INTRODUCTION

Software metrics is a standard to measure software or its specification. In recent years, quality of software is considered as a very important criterion. In order to improve the quality, we go with metrics calculation. Once the metrics is calculated we can alter the program accordingly. Metrics also helps to compare the program and check their performance. The software metrics are useful in representing the present state of the software and assist to associate and forecast the current accomplishment of software applications [8]. Metrics calculation is essential factor to obtain software with high quality and low testing and maintenance cost. Metrics can be used in different phases of software development life cycle like requirements phase, number of files, number of data elements, number of processes at specification phase, number of modules, fault statistics, module cohesion, module coupling etc. at design phase, total number of test cases, number of tests resulting in failure, total number of faults at implementation and integration phase, measure of effectiveness of inspections, measure of fault density at maintenance and inspection phase. The main objective of this paper is to evaluate different metrics at method and class level using a tool.

2. WHY USE TOOLS?

In manual calculation we may omit some value or interpret the data incorrectly. Using a tool means that the assessment is more repeatable and consistently calculated as the preconceived notion is removed. Example: assessing cyclomatic complexity or monitoring tools to assess the system behavior, static analysis tool for nesting the components. It helps to reduce human errors. Large calculations can be done easily. It displays the output in click of a button saving time. When we use a tool we can view the information easily by representing them in any graphical format as information presented in graphical format is much easier for human mind to interpret. In this project, a tool will be developed to calculate the metrics at class and method level and display the result in graphical format.

3. DIFFERENCE BETWEEN MANUAL AND AUTOMATED CALCULATION

The work humans do manually in calculating a metric will consume more time than the way we do it using an automated tool. For example, if we want to calculate Cyclomatic complexity manually, we have to draw flow charts and should count edges, nodes etc. Then by substituting in the formulae we get the result. But, if we measure the same complexity using the automated tool, we just need to provide the program and we will get the result in very less time compared to manual calculation. Mistakes can be reduced by using an automated tool and the use of automated tools to calculate metrics requires less knowledge and effort.

4. WHAT ARE WE MEASURING?

This tool calculates metrics based on the class level and method level. Metrics like number of lines, number of statements, number of branches, number of comments, maximum depth, percentage of branches, percentage of comments Cyclomatic complexity, coupling, depth of inheritance tree. In order to measure accurately, first, we have to know what we are measuring. We need measurement in all aspects of a software development. For example, in software development life cycle, we use metrics in all phases like measuring the requirements which changes in requirement phase, measuring the number of files, cost, duration and so on in specification phase, number of modules, Cyclomatic complexity in design phase, total number of test cases and number of tests resulting in failure in implementation and integration phase and faults detected in a page in maintenance phase.

5. METRICS DESCRIPTION

There are around 200 different metrics available [4] and metrics which this tool calculates are explained below with respect to the formula used to calculate them,

5.1 Lines of code

This metrics calculates the complexity of the software based on the length of the program. It is basis for measuring programmer productivity. i.e. higher the lines of code higher is the possibility of bug density. It counts the total lines in code including the blank space and comment lines. It gives a better assessment of project quality.

5.2 Number of comments

This metrics gives the number of single and multiline comments present in the source code. A good program must consist of 30%-75% of comments. If it is less than 30% then the program is considered as a poorly explained program and if comments exists more than 75% then the program file is considered as a document not a program.

5.3 Number of statements

This metric calculates the number of statements in the program. It is referred as a more robust method. This metric counts statement like break, continue, do, explicit constructor call, explicit super constructor call, for, if, return, switch, throw, try, and catch, finally, while, assignments. Method calls, pre/post increment/decrement.

5.4 Cyclomatic complexity

Code complexity associates with the defect rate and robustness of the application program. Code with good complexity helps in easier understanding and easier maintenance. It is used to measure the linearly independent path in a program. Cyclomatic complexity increases with number of decision path and loops. If the complexity is greater then there is more execution path which makes it difficult to understand and test. It is a testability metrics using which we can access the number of test case needed to test the program or module.

Cyclomatic complexity=Number of Decision Points+1

In this method, the code will be first converted into flow graph and then with the help of that flow graph complexity will be found using the formulae.

$$V(G) = E - N + 2$$

E=Number of edges of the graph and N=Number of nodes of the graph

Or we can use the formula; Where P is number of predicate nodes.

$$V(G) = P + 1$$

```
Public void oddeven()
{
int i=40;
while(i<20)
{
System.out.println("Value is",i);
if(i%2==0){
System.out.println("Even");
}
else{
System.out.println("Not eligible");
}
}
}
```

Figure-1. Code snippet.

The above snippet is converted into the flow graph as shown below,

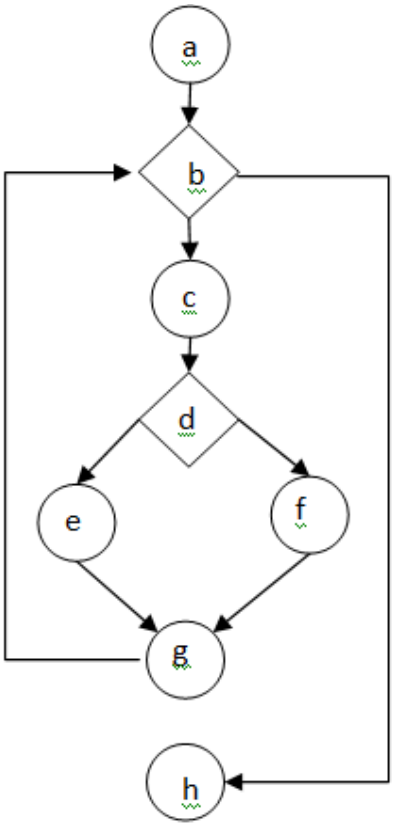


Figure-2. Flow graph.

After converting into flow graph, the Cyclomatic complexity is calculated as shown below.

Edges (E) = 9;

Nodes (N) = 8;

V(G) = E-N+2;

Therefore, V(G)= 9-8+2

V(G) = 3

(OR)

Number of predicate nodes (P) = 2;

V(G) = P+1;

Therefore, V(G) = 2+1

Figure-3. Metrics calculation.

If complexity is from

1 to 10 - The code is well written with high testability and low maintenance cost.

10-19 - The code is moderately complex with medium testability and medium maintenance cost or effort.

20-40 - The code is very complex with low testability and high maintenance cost or.

>40 - The code is not testable and any amount of money or effort to maintain the code may not be adequate.

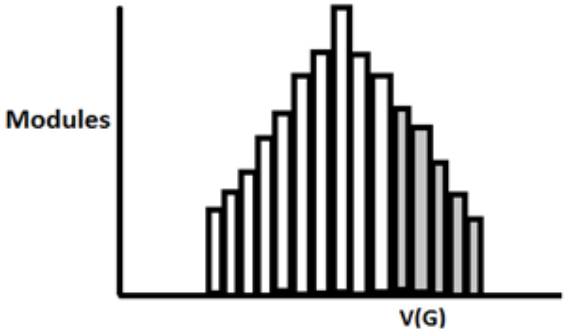
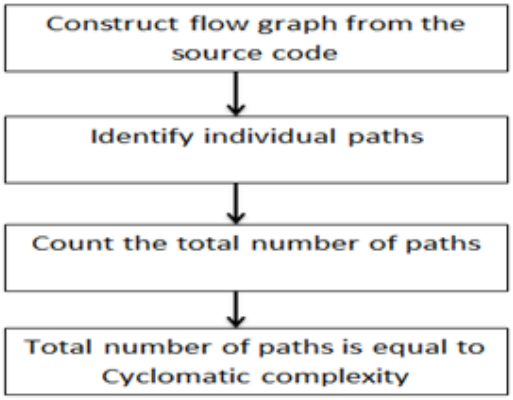


Figure-4. Cyclomatic complexity.

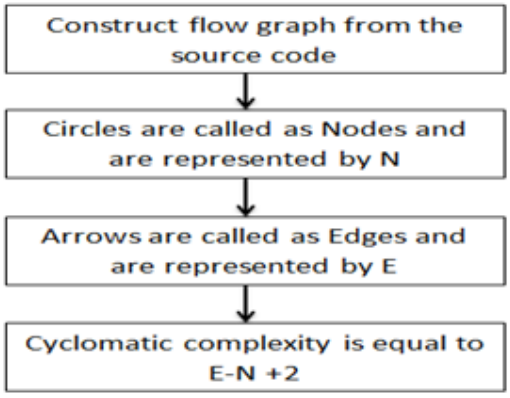
Cyclomatic complexity is referred as V(G). The above diagram clearly shows that the shaded region is considered to be more error prone region. Hence more the complexity, more the possibilities for error occurrence.

Algorithm for cyclomatic complexity

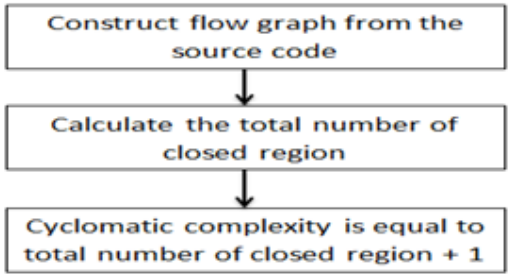
Method-1



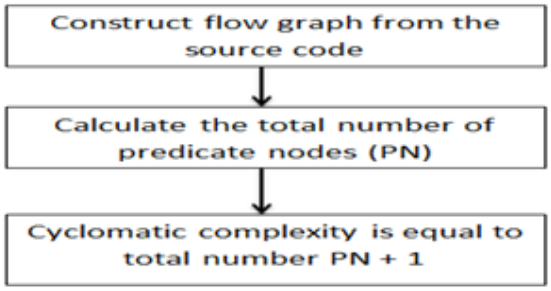
Method-2



Method 3:



Method 4:



5.5 CK metrics

CK metrics was developed by Chidamber and Kemerer. CK metrics is used to measure the design of object oriented programs.

CK metrics are better predictors than "the traditional" code metrics, which can only be collected at a later phase of the software development process [9]. The 6 metrics which comes under CK metrics suite are explained below,

Weighted method per class [WMC]: This metrics count the number of methods in for class. It helps to evaluate the time and effort needed to construct and maintain a class. The class which contains large number of methods is more application specific and hence it limits the possibility of reuse.

WMC = $\sum_{i=1}^{n} c_i$

C_1 is a class containing methods $m_1 \ldots m_2$ and complexity $c_1 \ldots c_2$. [12]

Depth of inheritance tree (DIT): This metrics counts the maximum number of steps from node to root of the tree. Complexity depends upon the number of steps (i.e. more the number of methods then more the complexity). If the class is deeper in the hierarchy then it is more complex to predict its behavior.

Number of children (NOC): This metrics counts of number of immediate subclasses. A class with many children requires more testing [7]. Larger the number of children, greater is the chance of misusing the subclasses.

Coupling between objects (CBO): This metrics calculates the number of other classes a class has coupled. Smaller the CBO value, higher the modularity and encapsulation. This coupling can occur via return types, field accesses, inheritance, arguments, method calls and exceptions.

Lack of cohesion of methods (LCOM): This metrics measures the divergence of methods in a class by means of instanced variables. This metrics will consider all the pairs of the class methods. Low cohesion increases the complexity and hence it is preferred to have a high ratio of cohesion. It does not provide encapsulation but helps in identifying low quality design.

Response for class (RFC): This metrics refers to set of methods which can be potentially executed with respect to the received message by object of that particular class [9]. It can also be defined as the number of methods in that class in addition to the number of methods called by those methods. Testing and maintenance becomes a complex process, if large numbers of methods are invoked from a class.

This tool calculates Weighted Methods per Class, Depth of Inheritance tree and Number of Children from the CK Metrics suite.

Table-1. Criteria level values.

Cost		Experience		Level	Benefits	Credibility	Validation	Relevance to reliability
Level	Value	Level	Value					
W	1.0	W	1.0	A	1.0	1.0	1.0	1
M	0.93	M	0.541	B	0.92	0.86	0.845	0.89
Q	0.7	L	0.23	C	0.59	0.65	0.449	0.789
Y	0.31	E	0.149	D	0.34	0.62	0.253	0.748
T	0	N	0	E	0.12	0.349	0	0.22
				F	0	0		0

6. MODULES DESCRIPTION

6.1 User interface

User interface defines a tool. The user interface should be easy to use, efficient and should provide the result in understandable format. The user interface of this tool contains two buttons in the home screen, namely find metrics and details.

Figure-5. Home page.

Find metrics button displays the file chooser dialog box as shown in Figure-6, let us to choose the java file for which metrics is to be calculated .

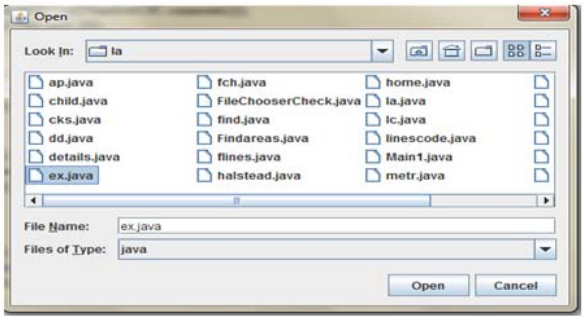


Figure-6. File chooser.

Details button displays the list of metrics, from which the user can select a particular metrics name button for which details will be displayed as shown in Figure-7.

Figure-7. Details of selected metrics.

6.2 Metrics calculation

This module calculates various metrics at method and class level based on formulae and integrate it with the user interface.

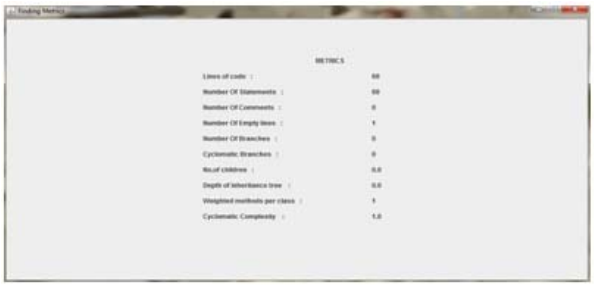


Figure-8. Calculated metrics values.

In Figure-8, the calculated metrics value of the program is displayed.

6.3 Chart generation

In this module pie chart will be generated for the calculated values of percentage of comments and percentage of branches.

In Figure-9, black color represents the percentage of comment lines in the given program and blue color represents the remaining lines of code.

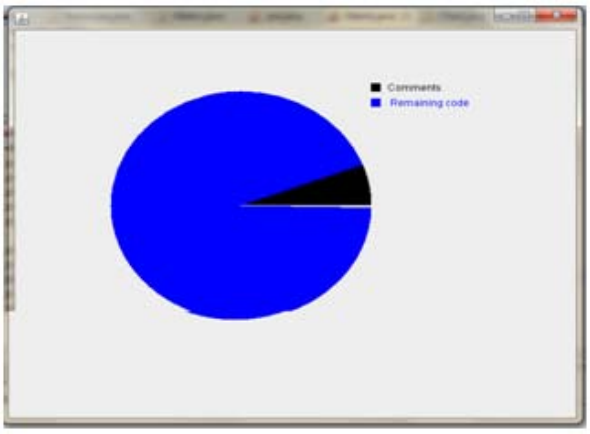


Figure-9. Percentage of comments.

In Figure-10, red color represents the percentage of branches in the given code and blue color represents the remaining lines of code.

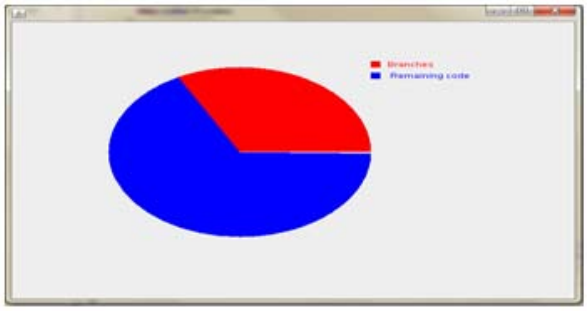


Figure-10. Percentage of branches.

The graphical representation helps us to understand the code better.

7. APPLICATION OF THIS TOOL

The purpose of using this tool is to save the user time and to improve their quality of software. This method will be used in software debugging, estimating cost, quality assurance testing, performance optimization of the software, and best personnel task assignments [6]. Software complexity measures are used to indicate whether the software has desirable attributes such as understandability, testability, maintainability and reliability. This tool will be widely used in java applications that require output to be displayed in graphical format. This tool also provides the basic information about the metrics being calculated. Hence the purpose of this tool is not only to calculate metrics but also to provide information about the metrics in order to help the user in understanding it still better.

8. CONCLUSION AND FUTURE WORK

This paper is discussed about the calculation of metrics at class and method level for the java program. In this tool, we display the output in graphical format to eliminate code smells and to maintain the threshold value. The source code of this software is freely accessible for everyone to download, develop and modify. This tool does not calculate metrics at system and package level which can be developed in future.

REFERENCES

[1] E.Umamaheswari, D.K. Ghosh. 2013. Developing a Reliability Prediction System Using Multivariate Analysis Theory on Software Quality Metrics, Vol. 3, Issue 1, IJETCSE.

[2] Radulescu. 2009. Open Source Tools to measure software complexity, IEEE.

[3] Tuhonglei. 2009. The Research on Software Metrics and Software Complexity Metrics, IEEE.

[4] Kaushal Bhatt, VinitTarey, Pushpraj Patel. 2012. Analysis of Source Lines of Code (SLOC) Metric, IEEE.

[5] Rüdiger Lincke, Jonas Lundberg and WelfLöwe, Comparing Software Metrics Tools, IEEE.

[6] Wikipedia http://en.wikipedia.org/wiki/Software_metric.

[7] ArtiChhikara and R.S.Chhillar, Analyzing the complexity of java programs using object oriented software metrics.

[8] Meyer B, Oriol M and Schoeller B. 2009. Software engineering: Lecture 17-18: estimation techniques and software metrics, Chair of software Engineering.

[9] Chidamber-Kemerer (ck) and Lorenze-kidd (lk) metrics to assess java programs, Jubair J. Al-Ja'afer and KhairEddin M. Sabri

[10] Software Quality metrics,zijian Yuan

[11] Hamer, P. G. and Frewin, G. D. 1982. M. H. Halstead's Software Science - A Critical Examination, In: Proceedings of the 6th International Conference on Software Engineering, Tokyo, Japan, pp. 197-206.

[12] 2001. An Overview of Object-Oriented Design Metrics, Daniel Rodriguez Rachel Harrison March.