

SOEN 6611: SOFTWARE MEASUREMENT

Milestone 3



|  |  |
| --- | --- |
| **Student ID** | **Name** |
| 40024381 | Amar Kumar |
| 40018973 | Charanpreet Kaur |
| 40013983 | Ajeeta Kaveti |
| 40021928 | Tarun Jeet Singh |
| 40019612 | Navnoor Multani |
| 40010983 | Lovish Tayal |
| 40031066 | Bitta Rani Rana |
| 40012480 | Mandeep Kaur |

Table of Contents

[Internal Metrics Implementation 1](#_Toc477729719)

[1. CK Metric Suite 1](#_Toc477729720)

[1. Weighted Method per Class (WMC) 1](#_Toc477729721)

[2. Depth of Inheritance Tree (DIT) 1](#_Toc477729722)

[3. Number of Children (NOC) 1](#_Toc477729723)

[4. Coupling between Objects (CBO) 2](#_Toc477729724)

[5. Response for Class (RFC) 2](#_Toc477729725)

[6. Lack of Cohesion of Methods (LCOM) 2](#_Toc477729726)

[2. McCabe’s Cyclomatic Complexity 3](#_Toc477729727)

[Challenges faced during the implementation: 4](#_Toc477729728)

[Testing Of Metrics 4](#_Toc477729729)

[Repository Link 6](#_Toc477729730)

[External Metrics Collection 6](#_Toc477729731)

[1. LOCMetric Tool: 6](#_Toc477729732)

[2. CK Metric Suite: 6](#_Toc477729733)

[3. Sourceforge: 6](#_Toc477729734)

[Analysis of JEdit 7](#_Toc477729735)

[Appendix 8](#_Toc477729736)

[Appendix 1: Internal Metrics Result 13](#_Toc477729737)

[Appendix 2: External Metrics Result 14](#_Toc477729738)

[LOCMetric 15](#_Toc477729739)

[Appendix 3: TEST SUITE 17](#_Toc477729740)

[References: 18](#_Toc477729741)

# Internal Metrics Implementation

## CK Metric Suite

### Weighted Method per Class (WMC)

WMC is the sum of complexity of the methods of the class. More the number of methods in a class, more will be the complexity and hence difficult to maintain. Classes with large WMC will be more fault-prone which leads to low quality. Therefore, low WMC values are desirable[1].

WMC = Number of Methods in the class

**Approach for implementation:**

In this class we are again overriding visit method by giving MethodDeclaration as a parameter to it. A method declaration is the union of a method declaration and a constructor declaration. This class works in a way that we check while, for, conditional do, enhanced for, switch case for the whole class and increases the complexity according to that.

### Depth of Inheritance Tree (DIT)

It is the maximum length of a path from class node to root node in the tree hierarchy. DIT measures the number of ancestor classes that can affect a class. The deeper the class in hierarchy, more methods or classes it will inherit, thus making it more complex. This increases the fault-proneness. Low values of DIT promote code reuse which is desirable.

DIT = maximum inheritance path from the class to the root class

**Approach for implementation:**

In this class we have calculated parent of a class by using method getSuperClass. Then we are checking whether parent is null or not. If parent is not null then we are using the method getQualifiedName to get the name of the parent. After that we are checking whether the parent name is ending with object if yes we are returning as every class has 1 parent and if not the we are incrementing DIT variable.

### Number of Children (NOC)

It measures the number of immediate sub-classes derived from the base class. High NOC indicates higher reuse and fewer faults and hence, it requires less testing.

NOC = number of immediate sub-classes of a class

**Approach for implementation:**

In NOC we overridden the visit method by passing the TypeDeclaration as a parameter which returns whether the node is a class type or a interface type and then we accessed the superclass of that node and then it verifies that class in the derived list of the classes available in our code if that has object super class which is the default class then it is not the child of any class and if it is then it simply increments in the NOC variable in the superclass calculations

### Coupling between Objects (CBO)

Two classes are said to be coupled if methods/instance variables declared or defined in one class are used by the other class. It is a bi-directional relationship. High coupling between classes is undesirable because it increases dependencies. The more independent a class is, the more reusable it is. Independent classes increase modularity. High CBO indicates more fault-proneness and hence, difficult to maintain [2].

CBO = number of classes to which a class is coupled

**Approach for implementation:**

This class is extending ASTVISITOR and we are overriding the visit method of that class. We are passing VariableDeclarationStatement as a parameter to this method. This kind of node collects several variable declaration fragments all sharing the same modifiers and base type.In a similar manner, we have used classes like ClassInstanceCreation, ArrayCreation, FieldDeclaration as a parameter to visit method and then we are calling coupleTo method in each visit method in order to check coupling.

### Response for Class (RFC)

It is the set of methods that can be executed in response to a message received by an object of that class. It measures the communication of a class with other classes. Higher values of RFC indicate more faults and hence difficult to maintain and understand [3].

RFC = M + R

M = number of methods in the class

R = number of methods directly called by methods of the class

**Approach for implementation:**

The response set of a class is a set of methods that can be potentially executed in response to a message received by an object of that class.In our class we first checked the number of methods by overriding the ASTVisitor's Visit method. In which we checked method invocation through passing MethodInvocation method and we also have considered the super class methods by passing SuperMethodInvocation and then count all the methods call and assigned them the RFC variable.

### Lack of Cohesion of Methods (LCOM)

It measures the cohesiveness of the class. Low cohesion indicates high complexity and more fault-proneness. Highly cohesive class promotes encapsulation which is desirable.

LCOM = P - Q, if P > Q

LCOM = 0, otherwise

Where, P is the set of method pairs which do not share attributes.

Q is the set of method pairs which share at least one attribute.

LCOM = 0 indicates a cohesive class.

LCOM > 0 indicates that the class needs or can be split into two or more classes.

**Approach for implementation:**

In LCOM we overridden the visit method by passing the FieldDeclaration and MethodDeclaration in the parameters and then we write another methods which checks where the declared field has been accessed by the method if it has been accessed by the method then increment variable which is holding the visited methods sum. In the calculating function we calculated the using the formula "|P|-|Q|" where |P| is the sum of methods which are not sharing any methods and |Q| is the sum of the methods which are sharing attributes.

## McCabe’s Cyclomatic Complexity

It is a software metric used to measure the complexity of a program. It measures the number of independent paths that can be executed through the source code. It is measured based on the control flow graph of the program. Higher values indicate higher complexity which increases testability and hence maintenance [5].

**Cyclomatic Complexity = E-N+2P**

Where, E = the number of edges in CFG

N = the number of nodes in CFG

P = the number of connected components in CFG

**Cyclomatic Complexity = D+1**

Where, D = the number of decision statements.

**Approach for implementation:**

In McCabe Cyclomatic Complexity we first read all the java files from the project directory and added their path in an array of string type. Then we passed each file to a function which is calculating cyclomatic complexity. That method first initialize a array with the allowable tokens that should be counted during calculating complexity. We have used "CC = d+1", where d is the number of decision statements. Afterwards it reads the code file and extracts the tokens from the code and after that it compares the extracted token with the allowable token and if it matched then it increments complexity variable. We have treated multiple conditions in If as different if condition.

# Challenges faced during the implementation:

The main challenge that we faced was the implementation of AST (Abstract Syntax Tree) parser.

**Solution proposed to overcome this challenge:**

We studied about the internal working of the AST parser, obtained information about the bindings in AST and how to access the nodes structural properties. We studied about the necessary classes that we have to use to implement AST parser and finally ended up in implementing AST successfully.

# 

# Testing Of Metrics

Testing of the implemented metrics is done by creating a test suite for CK metrics. First, we calculated the metric values manually and then compared the values using the implemented metrics. Correctness of implemented metrics is validated by comparing the results with manual implementation. UML diagram and CK test suite results are shown in *Appendix 3*. The result of metric implementation are shown in *Appendix 1 & Appendix 2.*

The test code for McCabe Cyclomatic Complexity is given below:

//Class 1

public class Test {

public void A()

{

int x = 1;

if (x == 1)

{

// Statement 1

}

else if (x == 2)

{

// Statement 2

}

else

{

// Statement 3

}

while (true)

{

// Statement 4

}

}

}

//Class 2

public class Test2 {

public void B()

{

int x = 10;

while (x == 1)

{

// Statement 5

}

}

}

Manual and automated computation reveal that the total cyclomatic complexity for the above code is 6.

CC1=D+1=3

CC2=D+1=3

**Total CC=6**

But we also compute the average cyclomatic complexity:

**Average CC=Total Complexity/Number of Classes = 6/2 = 3**

We have implemented both total and average complexity in the code.

# Repository Link

<https://github.com/Amarkumar93/SoftwareMeasurementProject>

This repository contains the entire code for implemented metrics and the test suite. The CSV files for Internal and External Metric Results are also available in the Document Directory of the repository. It also contains the final Milestone 3 document.

# 

# External Metrics Collection

It has been observed empirically that there exists a positive correlation between fault proneness and change-proneness which is directly linked to maintainability.

## LOCMetric Tool:

We used LOCMetric tool to calculate the external metrics of jEdit for 26 versions. LOCMetric provides several details of each version like Number of Directories, Source Files, and Comment Lines etc. More the number of directories, source files and comment lines, higher will be the risk of error and more change prone it will be. Newer versions of jEdit have more lines of code and hence more directories and source files. Therefore, maintainability of jEdit becomes higher as new versions are released.

## CK Metric Suite:

We also calculated Number of Methods and total Lines of Code for all the versions. jEdit 3.0.0 has 1680 NOM and 37306 LOC. But as more features are added to the software, NOM and LOC increase. jEdit 5.4.0 has 6959 NOM and 154019 LOC therefore, it is difficult to maintain. We calculated NOM and LOC from the implemented metric suite. High values of NOM and LOC are undesirable as it indicates more change-proneness and fault-proneness. Therefore, these values should be kept low to lower the maintenance effort.

## Sourceforge:

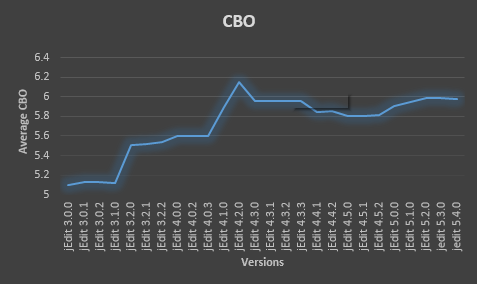
We used sourceforge to collect the bug report and change requests. Number of bugs metric lets us know about the total number of minor, major, severe bugs reported while development of various versions of sourceforge. More the Number of Bugs, lower will be the quality of the software and higher the maintainability. But as the bugs are reported and resolved, the quality improves. Number of changes reported are 244 which represent the change requests made during evolution of jEdit.

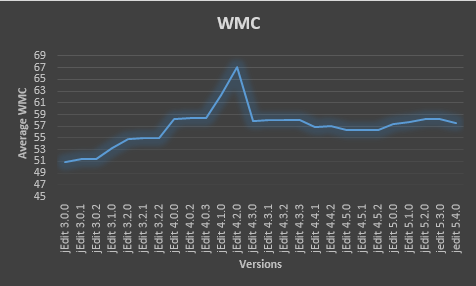
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Minor | Normal | Severe | Regressive | Changes |
| 775 | 1137 | 450 | 92 | 244 |

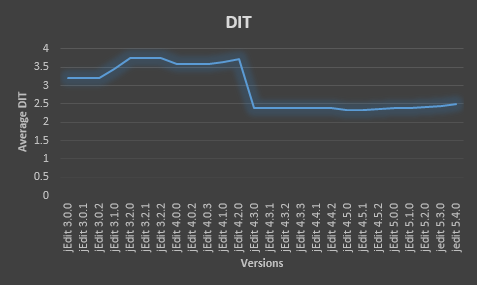
# Analysis of JEdit

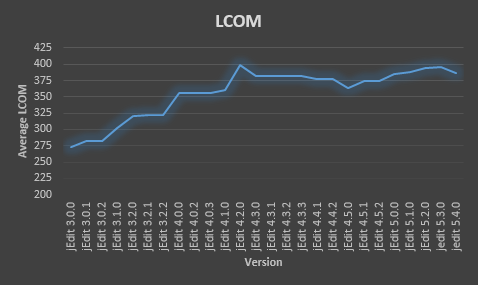
After analyzing all the versions of jEdit, we prove our hypothesis that the classes with poor CK and McCabe’s Cyclomatic Complexity are more fault-prone and change-prone, hence, difficult to maintain. The table given below represents the correlation between the complexity of different metrics calculated from our implemented metric suite and maintainability:

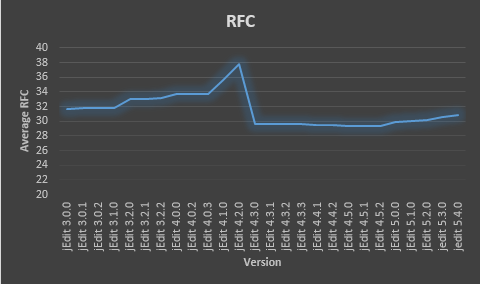
|  |  |  |  |
| --- | --- | --- | --- |
| **Metric Suite** | **Complexity** | **Maintainability** | **Desired Value of Metrics** |
| CBO | Increases | High | Low |
| WMC | Increases | High | Low |
| DIT | Decreases | Low | Low |
| NOC | Decreases | Low | Low |
| RFC | Decreases | Low | Low |
| LCOM | Increases | High | Low |
| McCabe's CC | Increases | High | Low |
| NOM | Increases | High | Low |
| LOC | Increases | High | Low |

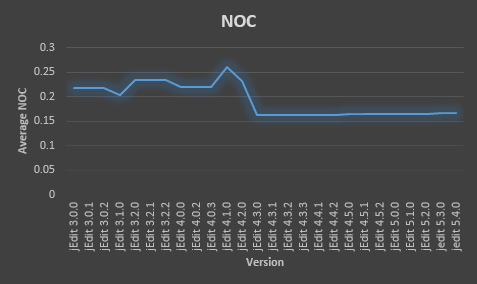


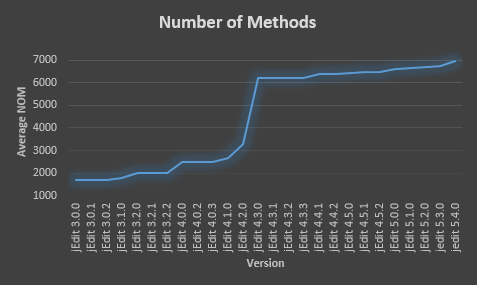


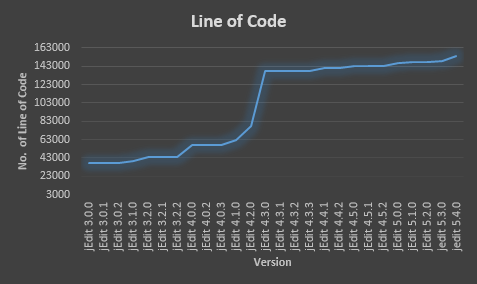


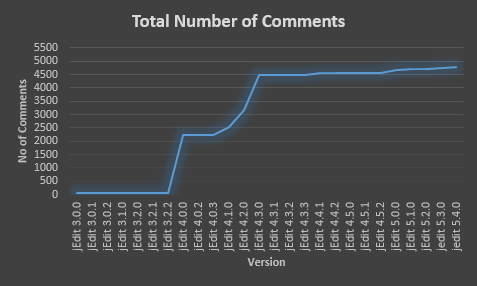


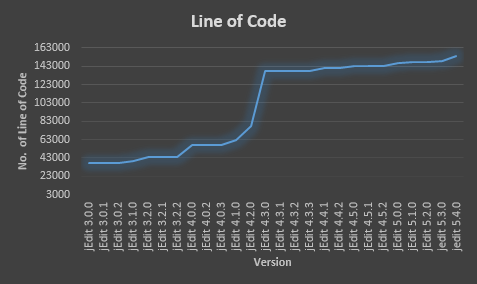












# Appendix

## Appendix 1: Internal Metrics Result

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Version** | **CK Metric Suite** | | | | | | **McCabe’s**  **Cyclomatic Complexity** |
| **CBO** | **WMC** | **DIT** | **NOC** | **RFC** | **LCOM** |
| jedit 5.4.0 | 5.976744 | 57.56395 | 2.505814 | 0.166667 | 30.88178 | 386.8372 | 27.44 |
| jedit 5.3.0 | 5.991968 | 58.2751 | 2.437751 | 0.166667 | 30.5261 | 395.6124 | 27.35 |
| jEdit 5.2.0 | 5.987903 | 58.19153 | 2.425403 | 0.165323 | 30.22379 | 394.1573 | 27.31 |
| jEdit 5.1.0 | 5.947581 | 57.66532 | 2.397177 | 0.165323 | 30.0504 | 387.9194 | 26.5 |
| jEdit 5.0.0 | 5.907258 | 57.32661 | 2.397177 | 0.165323 | 29.95363 | 385.2238 | 26.28 |
| jEdit 4.5.2 | 5.810101 | 56.38384 | 2.357576 | 0.165657 | 29.35152 | 374.497 | 25.58 |
| jEdit 4.5.1 | 5.806061 | 56.34747 | 2.345455 | 0.165657 | 29.33737 | 374.3737 | 25.56 |
| jEdit 4.5.0 | 5.80202 | 56.29697 | 2.345455 | 0.165657 | 29.30707 | 363.1475 | 25.49 |
| jEdit 4.4.2 | 5.85625 | 57.04583 | 2.402083 | 0.1625 | 29.51667 | 377.4688 | 25.53 |
| jEdit 4.4.1 | 5.85 | 56.94792 | 2.402083 | 0.1625 | 29.4625 | 376.8792 | 25.46 |
| jEdit 4.3.3 | 5.95914 | 58.03226 | 2.402151 | 0.163441 | 29.68387 | 382.5634 | 27.72 |
| jEdit 4.3.2 | 5.95914 | 58.0086 | 2.402151 | 0.163441 | 29.68387 | 382.5828 | 27.71 |
| jEdit 4.3.1 | 5.956989 | 57.98495 | 2.402151 | 0.163441 | 29.69032 | 381.4495 | 27.7 |
| jEdit 4.3.0 | 5.956989 | 57.95484 | 2.402151 | 0.163441 | 29.68172 | 381.314 | 25.68 |
| jEdit 4.2.0 | 6.153488 | 67.06977 | 3.72093 | 0.232558 | 37.77209 | 399.414 | 16.35 |
| jEdit 4.1.0 | 5.898396 | 62.27273 | 3.631016 | 0.262032 | 35.58824 | 360.2193 | 15.74 |
| jEdit 4.0.3 | 5.596685 | 58.36464 | 3.60221 | 0.220994 | 33.72376 | 355.9448 | 15.16 |
| jEdit 4.0.2 | 5.596685 | 58.35359 | 3.60221 | 0.220994 | 33.72376 | 355.9448 | 15.16 |
| jEdit 4.0.0 | 5.596685 | 58.27624 | 3.60221 | 0.220994 | 33.69613 | 356.1436 | 15.11 |
| jEdit 3.2.2 | 5.535948 | 55.07843 | 3.764706 | 0.235294 | 33.15686 | 321.5425 | 13.83 |
| jEdit 3.2.1 | 5.51634 | 54.91503 | 3.764706 | 0.235294 | 33.0719 | 321.4052 | 13.76 |
| jEdit 3.2.0 | 5.509804 | 54.82353 | 3.764706 | 0.235294 | 33.03922 | 320.8889 | 13.72 |
| jEdit 3.1.0 | 5.115942 | 53.21739 | 3.449275 | 0.202899 | 31.85507 | 302.4275 | 13.15 |
| jEdit 3.0.2 | 5.12782 | 51.45865 | 3.203008 | 0.218045 | 31.85714 | 282.7669 | 13.33 |
| jEdit 3.0.1 | 5.12782 | 51.44361 | 3.203008 | 0.218045 | 31.84962 | 282.7669 | 13.33 |
| jEdit 3.0.0 | 5.097744 | 50.94737 | 3.203008 | 0.218045 | 31.7218 | 273.0752 | 13.36 |

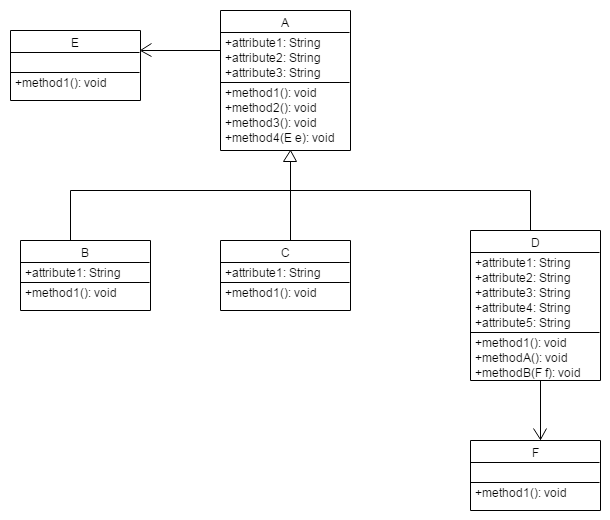
## Appendix 2: External Metrics Result

|  |  |  |
| --- | --- | --- |
| **Version** | **NOM** | **LOC** |
| jedit 5.4.0 | 6959 | 154019 |
| jedit 5.3.0 | 6744 | 149142 |
| jEdit 5.2.0 | 6710 | 148117 |
| jEdit 5.1.0 | 6661 | 147274 |
| jEdit 5.0.0 | 6625 | 146519 |
| jEdit 4.5.2 | 6467 | 143180 |
| jEdit 4.5.1 | 6462 | 143082 |
| jEdit 4.5.0 | 6439 | 142815 |
| jEdit 4.4.2 | 6393 | 140842 |
| jEdit 4.4.1 | 6387 | 140655 |
| jEdit 4.3.3 | 6227 | 137648 |
| jEdit 4.3.2 | 6226 | 137613 |
| jEdit 4.3.1 | 6220 | 137546 |
| jEdit 4.3.0 | 6218 | 137496 |
| jEdit 4.2.0 | 3306 | 77614 |
| jEdit 4.1.0 | 2675 | 62707 |
| jEdit 4.0.3 | 2483 | 56926 |
| jEdit 4.0.2 | 2483 | 56919 |
| jEdit 4.0.0 | 2481 | 56875 |
| jEdit 3.2.2 | 2001 | 43621 |
| jEdit 3.2.1 | 2001 | 43992 |
| jEdit 3.2.0 | 1999 | 43949 |
| jEdit 3.1.0 | 1775 | 39185 |
| jEdit 3.0.2 | 1705 | 37535 |
| jEdit 3.0.1 | 1705 | 37532 |
| jEdit 3.0.0 | 1680 | 37306 |

### LOCMetric

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Number of Directories | Source Files | Comment Lines |
| jedit 5.4.0 | 81 | 517 | 4787 |
| jedit 5.3.0 | 81 | 499 | 4737 |
| jEdit 5.2.0 | 81 | 497 | 4725 |
| jEdit 5.1.0 | 81 | 497 | 4699 |
| jEdit 5.0.0 | 81 | 497 | 4662 |
| jEdit 4.5.2 | 81 | 496 | 4554 |
| jEdit 4.5.1 | 81 | 496 | 4549 |
| jEdit 4.5.0 | 81 | 496 | 4543 |
| jEdit 4.4.2 | 80 | 481 | 4546 |
| jEdit 4.4.1 | 80 | 481 | 4544 |
| jEdit 4.3.3 | 79 | 465 | 4497 |
| jEdit 4.3.2 | 79 | 465 | 4497 |
| jEdit 4.3.1 | 79 | 465 | 4492 |
| jEdit 4.3.0 | 79 | 465 | 4490 |
| jEdit 4.2.0 | 17 | 215 | 3182 |
| jEdit 4.1.0 | 16 | 187 | 2533 |
| jEdit 4.0.3 | 15 | 181 | 2239 |
| jEdit 4.0.2 | 15 | 181 | 2239 |
| jEdit 4.0.0 | 15 | 181 | 2239 |
| jEdit 3.2.2 | 14 | 153 | 62 |
| jEdit 3.2.1 | 14 | 153 | 62 |
| jEdit 3.2.0 | 14 | 153 | 62 |
| jEdit 3.1.0 | 13 | 138 | 51 |
| jEdit 3.0.2 | 12 | 133 | 46 |
| jEdit 3.0.1 | 12 | 133 | 46 |
| jEdit 3.0.0 | 12 | 133 | 46 |

## Appendix 3: TEST SUITE



|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **ClassName** | **Type** | **CBO** | **WMC** | **DIT** | **NOC** | **RFC** | **LCOM** | **NOM** | **LOC** |
| TestSuite.A | class | 1 | 4 | 0 | 3 | 4 | 6 | 4 | 19 |
| TestSuite.B | class | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 7 |
| TestSuite.C | class | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 7 |
| TestSuite.D | class | 2 | 3 | 1 | 0 | 3 | 3 | 3 | 17 |
| TestSuite.E | class | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 6 |
| TestSuite.F | class | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 6 |

# References:

[1] Madhu Rohilla and P. K. Bhatia,” Prediction of Fault- Proneness Using CK Metrics,”in *International Journal of Emerging Technology and Advanced Engineering* (ISSN 2250-2459, ISO 9001:2008 Certified Journal, Volume 3, Issue 8, August 2013).

[2] Aman Kumar Sharma, Arvind Kalia, and Hardeep Singh, “Empirical Analysis of Object Oriented Quality Suites,” in *International Journal of Engineering and Advanced Technology (IJEAT) ISSN: 2249 – 8958*, *Volume-1, Issue-4*, April 2012.

[3] Chidamber & Kemerer object-oriented metrics suite [Online]. Available: [*http://www.aivosto.com/ project/help/pm-oo-ck.html*](http://www.aivosto.com/%20project/help/pm-oo-ck.html)

[4] <https://sourceforge.net/p/jedit/bugs> [Online].

[5] Tsantalis, Nikolaos. "Cyclomatic Complexity" 2013. Lecture. Available: https://moodle.concordia.ca/moodle/course/view.php?id=93707

[6] Tsantalis, Nikolaos. "The history of Cohesion Metrics" 2013. Lecture. Available: https://moodle.concordia.ca/moodle/course/view.php?id=93707