## *Assignment On*

***Object Oriented Structure Measurement***

***Software Metrics Lab***

***Couse Code: SE 3204***

## *Submitted To*


***Dipok Chandra Das***

***Assistant Professor***

***IIT, NSTU***


## Submitted by


**Md. Raju Biswas (ASH1925001M)**

**Sanzida Sultana (BKH1825010F)**

**Abdullah Al Mamun (ASH1925026M)**


5 March, 2023

# Contents

1 Introduction ..... 3
2 Coupling Metrics ..... 3
2.1 Coupling Between Object (CBO) ..... 3
2.1.1 Process of measurement ..... 3
Table 1: Table of finding the CBO ..... 4
2.2 Weighted Method Per Class ..... 5
Table 2: Table of weight giving ..... 5
Table 3: Table of finding the WMC ..... 5
2.2 Depth of Inheritance ..... 7
2.2.1 Process of measurement ..... 7
Table 4: Table of finding the DIT ..... 8
2.3 Number of Children (NOC) ..... 9
2.3.1 Process of measurement ..... 9
Table 5: Table of finding the NOC ..... 10
2.4 Response set For Class (RFC) ..... 11
2.4.1 Process of measurement ..... 11
Table 6: Table of finding the RFC ..... 11
3 Size Metrics ..... 12
3.1 Lines of Code (LOC): ..... 12
Table 7: Table of finding the LOC ..... 12
3.2 Number of Attributes ..... 12
3.2.1 Process of measurement ..... 12
Table 8: Table of finding the NA ..... 13
4 Cohesion Metrics ..... 14
4.1 Lack of Cohesion ..... 14
Table 9: Table of finding the LCOM ..... 14

# 1 Introduction

Object-oriented structure measurement metrics are used to assess the quality of the design and implementation of software systems that use object-oriented programming (OOP) concepts. The metrics provide insights into the complexity, maintainability, and reusability of the system's object-oriented structure, which can be used to identify areas for improvement and optimize system performance.

# 2 Coupling Metrics

Coupling metrics are used to measure the level of interdependence or connections between different modules, classes, or components within a software system. It indicates how much one module or class is dependent on another module or class. Coupling metrics are important in software development because high coupling can make a system more complex and difficult to maintain, test, and modify.

## 2.1 Coupling Between Object (CBO)

CBO (Coupling Between Objects) is a metric used to measure the degree of coupling or interdependence between classes in an object-oriented software system. It indicates the number of other classes that a particular class is coupled to. A high CBO value indicates that the class is highly dependent on other classes, making it more difficult to modify, test, and maintain.

### 2.1.1 Process of measurement

The process of measuring CBO (Coupling Between Objects) involves analyzing the relationships between classes in an object-oriented software system. The following steps can be taken to measure CBO:

1.Identify the classes in the software system “Coding_Helper” that need to be analyzed.

2. For each class, examine its instance variables, method arguments, and method return types to identify the other classes that it is coupled to. Note down the names of the coupled classes.

3.Count the number of different coupled classes for each class. This gives the CBO value for that class.

4.Repeat the process for all classes in the software system.

5.Analyze the CBO values to identify classes that are highly dependent on other classes. High CBO values indicate a high degree of coupling and may indicate areas of the system that are difficult to modify or maintain.

**Table 1: Table of finding the CBO**

| Classes | Value of CBO |
|---|---|
| Filereader.java | 2 |
| Filewriter.java | 4 |
| ProjectReader.java | 7 |
| Command.java | 18 |
| BoxAndWhiskerChart,java | 2 |
| CloneCheck.java | 5 |
| CosineSimilarity.java | 2 |
| Porter_stemmer | 5 |
| PreProcessing.java | 3 |
| TfIdfCalculate.java | 2 |
| getTf.java | 1 |
| Compress.java | 3 |
| CompressedFileWriteHelper.java | 4 |
| Decompress.java | 2 |
| DecompressedFileWriteHelper.java | 4 |
| FileContentReader.java | 1 |
| FileWriteHelper.java | 2 |
| FrequencyMapCreate.java | 1 |
| Huffman.java | 4 |
| HuffmanTuple.java | 4 |
| Node.java | 1 |
| mainDecode.java | 1 |
| mainEncode.java | 1 |
| Average_LOC.java | 2 |
| FileCount.java | 1 |
| LineOfCode.java | 1 |
| MethodCount.java | 3 |
| GrepContent.java | 2 |
| MethodFind.java | 3 |
| ProcessSearchFile.java | 3 |
| Search.java | 5 |
| Similarity.java | 3 |
| TfIdfCalculate.java | 2 |

### 2.2 Weighted Method Per Class

WMC stands for Weighted Methods per Class, which is a software metric used in object-oriented programming to measure the complexity of a class. WMC is calculated by counting the number of methods in a class and assigning a weight to each method based on its complexity. The weight is usually determined by the number of executable statements or the cyclomatic complexity of the method.

**Table 2: Table of weight giving**

| | **Method complexity category** | **Weights** |
|---|---|---|
| Cyclomatic complexity<2 | Simple | 3 |
| Cyclomatic complexity=2 | Medium | 4 |
| Cyclomatic complexity>2 | Complex | 5 |

**Table 3: Table of finding the WMC**

| Classes | Methods | Cyclomatic complexity | Weight |
|---|---|---|---|
| Filereader.java | fileEmpty(String path) | 3 | 5 |
| Filewriter.java | createProcessFile (String filename, String fileContent,String path) | 3 | 5 |
| ProjectReader.java | fileRead(String fullPath, int i) | 1 | 3 |
| | visitFile(Path f, BasicFileAttributes attr) | 2 | 4 |
| | getcurrentPath() | 3 | 5 |
| | getMethod(String path) | 1 | 3 |
| | LineCode(String Currentpath) | 2 | 4 |
| CloneCheck.java | pathGenerate(String projectName) | 3 | 5 |
| | getFileListforProject1(String projectOne) | 3 | 5 |
| | Code_clone(String project1, String project2) | 0 | 3 |
| CosineSimilarity.java | getCosinesimilarity() | 1 | 3 |
| | cosineSimilarity(double[] project1, double[] project2) | 1 | 3 |
| PreProcessing.java | ProcessFile(String filename, String content,String p) | 2 | 4 |
| | removePunctuation(String p) | 3 | 5 |
| | removeSpace(String fileAsString) | 3 | 5 |
| | removeKeyword(String fileAsString) | 1 | 3 |
| TfIdfCalculate.java | fileRead(String path) | 1 | 3 |
| | getUniqueWordProject1(String path1) | 1 | 3 |
| | IdfCal() | 2 | 4 |
| | tfIdfVectorProject1() | 0 | 3 |
| getTf.java | getTf(String[] fileContent, String term) | 1 | 3 |
| | getIdf(List allFile, String term) | 1 | 3 |
| Compress.java | Compress (String sourceFilePath, String targetFilePath) | 3 | 5 |

| | | | |
|---|---|---|---|
| | compressFile() | 3 | 5 |
| CompressedFileWriteHelper .java | CompressedFileWriteHelper(String path, ArrayList<HuffmanTuple> encodings) | 3 | 5 |
| | initMap(ArrayList<HuffmanTuple> encodings) | 1 | 3 |
| | writeBeginningOfFile(String headerString) | 1 | 3 |
| | writeEndOfFile() | 2 | 4 |
| | writeToFile() | 2 | 4 |
| | doWork(int currentByte) | 3 | 5 |
| DecompressedFileWriteHelper .java | DecompressedFileWriteHelper(String path) | 3 | 5 |
| | writeToFile() | 3 | 5 |
| | handleReadDictionary(int currentByte) | 2 | 4 |
| | handleReadFirstByte(int currentByte) | 3 | 5 |
| | handleDecodeByByte(int currentByte) | 1 | 3 |
| | doWork(int currentByte) | 1 | 3 |
| FileWriteHelper.java | FileWriteHelper(String path) | 2 | 4 |
| | initFile() | 2 | 4 |
| FrequencyMapCreate.java | FrequencyMapCreate() | 3 | 5 |
| | doWork(int currentByte) | 2 | 4 |
| Huffman.java | huffman(Map<Character, Integer> map) | 1 | 3 |
| | performInorderTraversal() | 1 | 3 |
| | convertMapToList(Map<Character, Integer> map | 1 | 3 |
| | createMapFromFile(String filePath) | 2 | 4 |
| | canonizeHuffmanTree(Node root) | 2 | 4 |
| | sortHuffmanTuples(ArrayList<HuffmanTuple> list) | 2 | 4 |
| | generateLookupCode(ArrayList<HuffmanTuple> encodings) | 2 | 4 |
| HuffmanTuple.java | HuffmanTuple(char letter, String representation) | 2 | 4 |
| | toString() | 2 | 4 |
| Node.java | Node(char letter, int freq) | 3 | 5 |
| | compareTo(Node o) | 3 | 5 |
| GrepContent.java | findBetweenBraces(int start, String fileContent) | 0 | 3 |
| | getLineNumber(String word, String file, int preLine) | 2 | 3 |
| MethodFind.java | getMethod(String filename, String fileContent) | 1 | 3 |
| | getConstructor(String filename, String fileContent) | 2 | 4 |
| ProcessSearchFile.java | processMethod(String filename, String fileContent) | 1 | 3 |
| | queryProcess(String query) | 2 | 4 |
| | breakWord(String fileAsString) | 1 | 3 |
| Search.java | visitFile(Path f, BasicFileAttributes attr) | 2 | 4 |
| | processProject(String projectpath, String Projectname) | 3 | 5 |
| | getProcessFilepath(String projectname) | 3 | 5 |
| | getProjectFile(String projectname) | 3 | 5 |
| | getFile(String projectPath, String fileName) | 0 | 3 |
| | SearchingResult(String query, String projectPath) | 2 | 4 |
| Similarity.java | getCosine() | 2 | 4 |
| | getResult() | 3 | 5 |

| | | | |
|---|---|---|---|
| | Idfcal() | 1 | 3 |
| | UniqueQueryTerms(String processQuery) | 1 | 3 |
| | ProjectTfIdfCal() | 3 | 5 |
| | queryTfIdfCal(String processQuery) | 2 | 4 |

Weighted method per class = (total weighted method / No of classes )

= ( 252 / 33) = 7.63

## 2.2 Depth of Inheritance

DIT stands for Depth of Inheritance Tree, which is a metric used to measure the depth of inheritance of a class within an inheritance hierarchy. The DIT value for a class is determined by counting the number of steps from the class to the root of the inheritance tree.

### 2.2.1 Process of measurement

Here is the process to calculate DIT for a class

1.Identify the class for which you

want to calculate the DIT.

2.Draw the inheritance hierarchy diagram that includes the class.

3.Count the number of steps from the class to the root of the inheritance tree.

4.The number of steps counted in step 3 is the DIT value for the class.

**Table 4: Table of finding the DIT**

| Classes | DIT |
|---|---|
| Filereader.java | 0 |
| Filewriter.java | 0 |
| ProjectReader.java | 0 |
| Command.java | 0 |
| BoxAndWhiskerChart,java | 0 |
| CloneCheck.java | 0 |
| CosineSimilarity.java | 0 |
| Porter_stemmer | 0 |
| PreProcessing.java | 0 |
| TfIdfCalculate.java | 0 |
| getTf.java | 0 |
| Compress.java | 0 |
| CompressedFileWriteHelper.java | 0 |
| Decompress.java | 0 |
| DecompressedFileWriteHelper.java | 1 |
| FileContentReader.java | 0 |
| FileWriteHelper.java | 1 |
| FrequencyMapCreate.java | 1 |
| Huffman.java | 0 |
| HuffmanTuple.java | 0 |
| Node.java | 1 |
| mainDecode.java | 0 |
| mainEncode.java | 0 |
| Average_LOC.java | 0 |
| FileCount.java | 0 |
| LineOfCode.java | 0 |
| MethodCount.java | 0 |
| GrepContent.java | 0 |
| MethodFind.java | 0 |
| ProcessSearchFile.java | 0 |
| Search.java | 0 |
| Similarity.java | 0 |
| TfIdfCalculate.java | 0 |

### 2.3 Number of Children (NOC)

NOC stands for Number of Children. It is a software metric that measures the number of immediate subclasses of a class. It is used to measure the degree of coupling between a class and its subclasses. NOC can help in understanding the complexity of the class hierarchy and identifying classes that may need refactoring to improve maintainability.

#### 2.3.1 Process of measurement

The process of measuring the Number of Children (NOC) metric involves counting the number of direct subclasses that inherit from a given class. To measure NOC, follow these steps:

1.Identify the class for which you want to measure NOC.

2.Count the number of direct subclasses that inherit from the identified class.

3.Assign the counted number as the NOC value for the identified class.

It is important to note that NOC only considers the immediate subclasses of a class and not any indirect subclasses.

**Table 5: Table of finding the NOC**

| Classes | NOC |
|---|---|
| Filereader.java | 0 |
| Filewriter.java | 0 |
| ProjectReader.java | 0 |
| Command.java | 0 |
| BoxAndWhiskerChart,java | 0 |
| CloneCheck.java | 0 |
| CosineSimilarity.java | 0 |
| Porter_stemmer | 0 |
| PreProcessing.java | 0 |
| TfIdfCalculate.java | 0 |
| getTf.java | 0 |
| Compress.java | 0 |
| CompressedFileWriteHelper.java | 0 |
| Decompress.java | 0 |
| DecompressedFileWriteHelper.java | 0 |
| FileContentReader.java | 2 |
| FileWriteHelper.java | 2 |
| FrequencyMapCreate.java | 1 |
| Huffman.java | 0 |
| HuffmanTuple.java | 0 |
| Node.java | 0 |
| mainDecode.java | 0 |
| mainEncode.java | 0 |
| Average_LOC.java | 0 |
| FileCount.java | 0 |
| LineOfCode.java | 0 |
| MethodCount.java | 0 |
| GrepContent.java | 0 |
| MethodFind.java | 0 |
| ProcessSearchFile.java | 0 |
| Search.java | 0 |
| Similarity.java | 0 |
| TfIdfCalculate.java | 0 |

## 2.4 Response set For Class (RFC)

Response for a Class (RFC) is a software metric used to measure the number of methods in a class that can be executed in response to a message received by an object of that class. It is a measure of the size and complexity of a class and can be used to assess the potential maintainability and reusability of the class.

### 2.4.1 Process of measurement

RFC is the number of local methods plus the number of external methods called by local methods.

**Table 6: Table of finding the RFC**

| Classes | Number of local methods | External methods called by local methods | RFC |
|---|---|---|---|
| Filereader.java | 1 | 0 | 1 |
| Filewriter.java | 1 | 0 | 1 |
| ProjectReader.java | 3 | 0 | 3 |
| Command.java | 18 | 5 | 23 |
| BoxAndWhiskerChart.java | 3 | 0 | 3 |
| CloneCheck.java | 4 | 9 | 13 |
| CosineSimilarity.java | 2 | 0 | 2 |
| Porter_stemmer | 14 | 0 | 14 |
| PreProcessing.java | 4 | 2 | 6 |
| TfIdfCalculate.java | 6 | 0 | 6 |
| getTf.java | 2 | 0 | 2 |
| Compress.java | 2 | 3 | 5 |
| CompressedFileWriteHelper.java | 6 | 2 | 8 |
| Decompress.java | 4 | 1 | 5 |
| DecompressedFileWriteHelper.java | 6 | 5 | 11 |
| FileContentReader.java | 1 | 0 | 1 |
| FileWriteHelper.java | 3 | 0 | 3 |
| FrequencyMapCreate.java | 2 | 0 | 2 |
| Huffman.java | 15 | 2 | 17 |
| HuffmanTuple.java | 2 | 0 | 2 |
| Node.java | 5 | 0 | 5 |
| mainDecode.java | 4 | 2 | 6 |
| mainEncode.java | 4 | 1 | 5 |
| Average_LOC.java | 1 | 0 | 1 |
| FileCount.java | 1 | 0 | 1 |

| | | | |
|---|---|---|---|
| LineOfCode.java | 1 | 0 | 1 |
| MethodCount.java | 1 | 2 | 3 |
| GrepContent.java | 2 | 0 | 2 |
| MethodFind.java | 2 | 3 | 5 |
| ProcessSearchFile.java | 3 | 6 | 9 |
| Search.java | 5 | 12 | 17 |
| Similarity.java | 2 | 1 | 3 |
| TfIdfCalculate.java | 5 | 1 | 6 |

# 3 Size Metrics

Size metrics are used to measure the size of a software system or component, typically in terms of lines of code or other similar measures. Here are some common size metrics:

## 3.1 Lines of Code (LOC):

**Table 7: Table of finding the LOC**

| Project Name | Packege Name | Files | Line Of code |
|---|---|---|---|
| | IO | 3 | 125 |
| | Code_Clone | 7 | 892 |
| | console | 1 | 475 |
| | huffman | 12 | 842 |
| | metrics | 4 | 108 |
| | searching | 6 | 573 |
| Total | 6 | 33 | 3016 |

## 3.2 Number of Attributes

To determine the number of attributes of a given class in Java, we need to count the number of member variables or fields declared within the class. These member variables can be either instance variables (also known as non-static fields) or static variables.

### 3.2.1 Process of measurement

One such size metric is the number of attributes of a class, which is calculated as follows:

1. Count the number of instance variables (also known as non-static fields) declared within the class.

2. Count the number of static variables declared within the class.

3. Add the counts from step 1 and step 2 together to get the total number of attributes of the class.

**Table 8: Table of finding the NA**

| Classes | NA |
|---|---|
| Filereader.java | 0 |
| Filewriter.java | 0 |
| ProjectReader.java | 5 |
| Command.java | 4 |
| BoxAndWhiskerChart,java | 2 |
| CloneCheck.java | 4 |
| CosineSimilarity.java | 1 |
| Porter_stemmer | 0 |
| PreProcessing.java | 0 |
| TfIdfCalculate.java | 9 |
| getTf.java | 0 |
| Compress.java | 2 |
| CompressedFileWriteHelper.java | 1 |
| Decompress.java | 2 |
| DecompressedFileWriteHelper.java | 9 |
| FileContentReader.java | 0 |
| FileWriteHelper.java | 4 |
| FrequencyMapCreate.java | 1 |
| Huffman.java | 1 |
| HuffmanTuple.java | 2 |
| Node.java | 4 |
| mainDecode.java | 0 |
| mainEncode.java | 1 |
| Average_LOC.java | 0 |
| FileCount.java | 1 |
| LineOfCode.java | 1 |
| MethodCount.java | 0 |
| GrepContent.java | 0 |
| MethodFind.java | 0 |
| ProcessSearchFile.java | 0 |
| Search.java | 1 |
| Similarity.java | 1 |
| TfIdfCalculate.java | 7 |

# 4 Cohesion Metrics

Cohesion metrics are used to measure how well a class or module is focused on a single purpose or responsibility. There are several cohesion metrics that can be used to evaluate the quality of a software design:

Example: LCOM

## 4.1 Lack of Cohesion

Process of measurement:

Process of measurement:
LCOM=1- (sum (MF)/M*F)

where, M is the number of methods in a class
F is the number of instance field in the class
MF is the number of methods of the class accessing a particular instance field
sum(MF) is the sum of MF over all instance fields of the class

**Table 9: Table of finding the LCOM**

| Classes | Number of methods | Number of instance field | LCOM |
|---|---|---|---|
| Filereader.java | 1 | 1 | 0 |
| Filewriter.java | 1 | 0 | 1 |
| ProjectReader.java | 3 | 5 | 0.66 |
| Command.java | 18 | 4 | 0.82 |
| BoxAndWhiskerChart,java | 3 | 2 | 0.66 |
| CloneCheck.java | 4 | 4 | 0.5 |
| CosineSimilarity.java | 2 | 1 | 0.5 |
| Porter_stemmer | 14 | 0 | 1 |
| PreProcessing.java | 4 | 0 | 1 |
| TfIdfCalculate.java | 6 | 9 | 0.63 |
| getTf.java | 2 | 0 | 1 |
| Compress.java | 2 | 2 | 0 |
| CompressedFileWriteHelper.java | 6 | 1 | 0.33 |
| Decompress.java | 4 | 2 | 0.5 |
| DecompressedFileWriteHelper.java | 6 | 7 | 0.5 |
| FileContentReader.java | 1 | 0 | 1 |
| FileWriteHelper.java | 3 | 4 | 0.66 |
| FrequencyMapCreate.java | 2 | 1 | 0 |

| | | | |
|---|---|---|---|
| Huffman.java | 15 | 0 | 1 |
| HuffmanTuple.java | 2 | 2 | 0 |
| Node.java | 5 | 5 | 0.64 |
| mainDecode.java | 4 | 0 | 1 |
| mainEncode.java | 4 | 1 | 0 |
| Average_LOC.java | 1 | 0 | 1 |
| FileCount.java | 1 | 0 | |
| LineOfCode.java | 1 | 1 | 0 |
| MethodCount.java | 1 | 0 | 1 |
| GrepContent.java | 2 | 0 | 1 |
| MethodFind.java | 2 | 0 | 1 |
| ProcessSearchFile.java | 3 | 0 | 1 |
| Search.java | 5 | 1 | 0.6 |
| Similarity.java | 2 | 1 | 0.5 |
| TfIdfCalculate.java | 5 | 7 | 0.57 |