

SOEN 6611: SOFTWARE MEASUREMENT

Milestone 1



|  |  |
| --- | --- |
| **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**

[1. Team Formation 1](#_Toc474530400)

[2. Type of the study 1](#_Toc474530401)

[3. Related studies 1](#_Toc474530402)

[4. Projects 2](#_Toc474530403)

[5. Metrics 3](#_Toc474530404)

[1. MOOD Metrics: 3](#_Toc474530405)

[2. Chidamber & Kemerer Object-Oriented Metrics Suite 5](#_Toc474530406)

[3. Lorenz and Kidd’s Metrics 5](#_Toc474530407)

[4. QMOOD Metrics: 6](#_Toc474530408)

[5. Cyclomatic Complexity 7](#_Toc474530409)

[6. Resource Planning 8](#_Toc474530410)

[7. References 9](#_Toc474530411)

# Team Formation

The table below represents the roles and responsibilities of various team members:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Research Analyst** | **Programming** | **Paper Writing** | **Statistical Analysis** | **Presentation** |
| Charanpreet Kaur | Lovish Tayal | Charanpreet Kaur | Ajeeta kaveti | Charanpreet Kaur |
| Amar Kumar | Amar Kumar | Ajeeta Kaveti | Bitta Rani Rana | Navnoor Multani |
| Tarun Jeet Singh | Tarun Jeet Singh | Mandeep Kaur | Amar Kumar | Lovish Tayal |
| Navnoor Multani | Navnoor Multani | Bitta Rani Rana | Tarun Jeet Singh | Mandeep Kaur |
| Ajeeta Kaveti | Bitta Rani Rana | Lovish Tayal | Mandeep Kaur |  |

# 

# Type of the study

Our research will be based on Correlation of metrics with other software artifacts/activities. We have made the following hypothesis:

* Classes with poor MOOD and CK metric values are more fault-prone, change prone, and tested more frequently. Hence, difficult to maintain.
* Classes with high cyclomatic complexity are more error-prone and difficult to test and need more time for maintenance.

# Related studies

Madhu Rohilla et al. [1] conducted a study to describe how Chidamber and Kemerer’s object-oriented metric suite is useful to illustrate fault-proneness of the system. Various hypothetical examples, to access the applicability CK metric suite (WMC, DIT, CBO, RFC, NOC, and LCOM) to predict the bugs in classes, are used. The empirical results suggest that DIT metric is the best metric to predict the fault-proneness of classes and it is the most useful to improve the quality and reliability of the design. Aman Kumar Sharma et al. [2] conducted a study to empirically validate the Chidamber and Kemerer (CK) Metrics and Metrics for Object-Oriented Design (MOOD) Metric Suite across three latest versions of JfreeChart software to predict its quality. The results of this study will provide a perception for the relationship between object oriented design metrics and the quality of the software. Ritu Chauhan et al. [3] investigated that the main goal of software quality estimation is to identify and minimize error-prone tasks to reduce development cost.In this research, they have analyzed QMOOD, CK and MOOD metrics and for the case studies, they have used JAVA RMI classes and subclasses to determine the impact of different metrics. The main conclusion of the study is that it is very important to measure software quality early so that by using different metrics like QMOOD, MOOD, and CK we can estimate software quality and can improve it to make the best system. Jubair Al-Ja’afer et al. [4] investigated that the problem statement is to assess the quality of the software system quantitatively and objectively using object-oriented metrics. We have learned from this study that we can analyze our project results obtained from the different releases to figure out how the program quality is varying in term of error-proneness and maintainability. Dr Sonal Chawla et al. [5] investigated the complexity of the software system to help in better maintainability of the software code, retaining its quality and reducing the software maintenance cost. The values of the software metrics have been used to determine the complexity and maintainability of the code. It has been concluded that only object-oriented metrics allow the modifications to make it cost effective and improve its quality. Jeenam Chawla et al. [6] emphasized on finding ways to predict the quality attributes of the software or system by analyzing the object oriented metrics. Main focus is given to object oriented metrics given by CK in order to correctly find out the fault-proneness of the system. Class level object oriented metrics was used to perform the investigation. It’s revealed that CBO metric is best in predicting fault proneness. However, LOC metric is suitable for quick fault prediction. The study suggests that one can make use of metrics as means to efficiently define the quality of a system and relate them to economic variables such as productivity, rework effort, etc. Mr. U. L. Kulkarni et al. [7] conducted a study on six java based open source software systems to validate CK metrics to evaluate the quality of the system and to find probable design faults at both architectural and component level. The result of the study indicates that low values of DIT and NOC in the system implies improper use of inheritance. In contrast to it, CBO values close to 2 is not a problem. From the study, we have learnt that WMC, LCOM and CBO are related to each other and high values of these 3 increases complexity of the system.

# Projects

* 1. **Selected Version**

**jEdit:** It is a programmer’s text editor written in Java. It is customizable with plugins. It uses the Swing toolkit for the GUI. It can be configured as a rather powerful IDE through the use of its plugin architecture. It has 39 versions.

* 1. **Additional Information:**
* We will be working on first-five versions, middle-five versions, and last-five versions of each of the projects. This range of versions will help in comprehensive research and study of the metrics to get more optimized results.
* To compile the project in Eclipse, we first imported it. There were few libraries that were missing in the source code so we imported those libraries first. It was compiled successfully without any error except with some warnings.
  1. **Tools:** Following are the tools that we will be using. But a few of them might be added or removed as we proceed in our work.
* Eclipse
* JDeodorant Plugin
* Class Dependency Analyzer
* MS Project
* MS PowerPoint
* MS Word
* MS Excel etc.

# Metrics

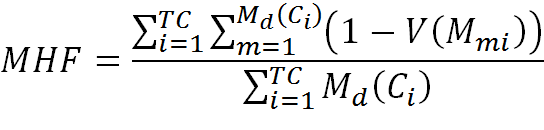
List the set of metrics you are planning to implement and use in your study. Include references to papers that contain a formal definition for the selected metrics.

## MOOD Metrics:

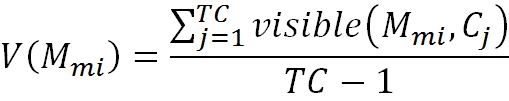
The MOOD metrics are designed to provide the quality of the Object Oriented project. The main factors which were taken into account in this metrics are encapsulation, Inheritance, Polymorphism and coupling. There are six metrics which are calculated using those object oriented paradigm.

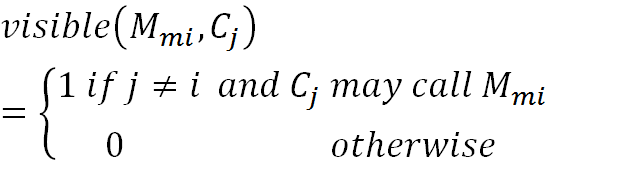
* 1. **Method Hiding Factor**

Method Hiding Factor measures how methods are encapsulated in the class. A low MHF value indicates that large number of methods are unprotected and high MHF indicated that the design has encapsulated very large number of methods.



Where Md(Ci) is the number of methods declared in class Ci , TC is the total number of classes, and



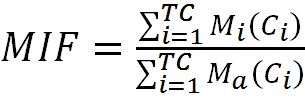
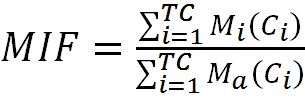


* 1. **Attribute Hiding Factor**

Attribute Hiding Factor measures how attributes are encapsulated in the class. AHF has the same definition as MHF. AHF – 100% is the ideal value.

* 1. **Method Inheritance Factor**

The method Inheritance factor can be defined as follows:

 , where

*Mi*(*Ci*) is the number of methods inherited (and not overridden) in class *Ci*.

*Md*(*Ci*) is the number of methods actually declared in class *Ci*

 + 

MIF low value indicates lack of inheritance and high value indicates that most of the subclasses are not declaring their own methods.

* 1. **Attribute Inheritance Factor**

It is same as MIF. But using attributes instead of methods.

* 1. **Polymorphism Factor**

It is the measure of the degree of method overriding in the class inheritance tree.

**PF = Number of method overrides / maximum number of possible method overrides**

PF high value indicates that all methods are overridden in all derived class and low value indicates that the project uses classes or inheritance.

* 1. **Coupling Factor**

Coupling Factor measures the actual couplings among classes in relation to the maximum number of possible couplings.

**CF = Actual Couplings / Maximum Possible Couplings**

CF high values indicates that all classes are coupled to other classes and low values indicates that no classes are coupled

## Chidamber & Kemerer Object-Oriented Metrics Suite

C&K metrics are used to measure design complexity and to study the impact on quality attributes such as maintainability etc.

1. **Weighted Methods per Class:** It is the sum of complexities of its methods. High WMC means more faults and classes with more methods limit the possibility of reuse.

The Weighted Methods per Class is defined as follows:

Where c1…cn be the complexity of the methods of a class with methods M1, ...,Mn.

If all method complexities are considered to be unity, the WMC = n, the number of methods.

1. **Depth of Inheritance Tree:** It is the maximum length from node to root of the tree. Deeper a class in hierarchy, more complex it will be because the class will inherit more methods. High DIT means more faults. It has been found that the most fault-prone classes are the ones in the middle of the tree. A recommended DIT is 5 or less.
2. **Number of Children:** It is the number of immediate subclasses of a class in class hierarchy. High NOC means high reuse of base class and it may require more testing, therefore fewer faults.
3. **Coupling between Objects:** It represents the number of classes coupled to a given classes. Classes can be coupled through method calls, inheritance etc. High CBO is undesirable as it prevents reuse. High CBO also means that it will be more prone to faults.
4. **Response for a Class:** It measures the number of methods that will be executed when an object of that class receives a message. High RFC indicates more faults which makes testing complicated.

RFC = M + R (First-step measure)

RFC’ = M + R’ (Full measure)

M = number of methods in the class

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

R’ = number of remote methods called, recursively through the entire call tree

## Lorenz and Kidd’s Metrics

The metric counts the total number of class variables in a class. These metrics are classified into: Class size metrics, Class inheritance metrics and Class Internals metrics

1. **Class size metrics**

**NIM:** This metric counts all public, protected and private methods defined for the class.

**NCM:** This metric counts the total number of class methods in a class.

**NCV:** The metric counts the total number of class variables in a class.

1. **Class inheritance metrics**

**SIX: (**Number of overridden methods\* hierarchy nesting level)/Total no. of methods

**NMA:** This metric counts the total number of methods defined in a subclass

1. **Class internals**

**APPM:** It is abbreviated as Average Parameters per Method metric.

**APPM = Total of methods parameters / Total number of methods**

## QMOOD Metrics:

The following are the metrics we have used:

1. **Design Size In Class (DSC)**

The total number of classes counted in the design; typically by counting the elements contained within. For example the number of operation in the class, number of classes in a package. It correspond to the object-oriented design property of design class in QMOOD.

DSC is an ideal metrics for developing cost or effort estimation for maintenance activities. Input for project planning purposes are usually done using the estimates developed.

Empirical studies have confirmed the importance of DSC as the main cost driver in a software Project.

1. **Number of Polymorphic Methods (NOP):**

The number of methods counted that can exhibit polymorphic behavior. It can be interpreted as the total sum overall classes in which a method can be exhibit polymorphic behavior if it is overridden by one or more descendent classes.

1. **Data Access Metric (DAM):**

The Data Access metric is the ratio of the number of protected attributes to the total number of attributes declared in the class. It is the average of all design classes corresponding to at least one attribute, of the ratio of private to the total attributes in a class.

1. **Direct Class Coupling (DCC):**

The Direct Class Coupling metric is a count of the different number of classes that a class is directly related to. The metric includes classes that are directly related by attribute declarations and message passing parameters in methods.

It is an average over all classes when applied to a design as a whole. The method parameter or attribute type gives the number of distinct user defined classes.

1. **Measure of Functional Abstraction (MFA):**

The Measure of Functional Abstraction metric is the ratio of the number of methods inherited by a class to the total number of methods accessible by member methods of the class.

1. **Cohesion Among Methods of Class (CAM):**

The Cohesion among Methods of Class metric computes the relatedness among methods of a class based upon the parameter list of methods. The metric is computed using the summation of the intersection of parameters of a method with the maximum independent set of all parameter types in the class. It is an average over all classes with at least one methods.

## Cyclomatic Complexity

It is used to measure minimum number of possible paths through source code. We are using this metric because Cyclomatic complexity can be used in Object Oriented programs at method level.

Cyclomatic complexity= E-N+2P where

E= number of edges in flow graph

N= number of nodes in flow graph

P= number of nodes that have exit points

Programs with Cyclomatic complexity of

* 1-4 are considered good
* 5-7 are considered ok
* 8-10 are considered for re-factoring
* 11+ means refactor now

# C:\Users\n_multa\Desktop\Gantt Chart.pngResource Planning

# 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. Ritu Chauhan,Rahul Singh, and Ashish Saraswat, “Estimation of Software Quality using Object Oriented Design Metrics,” in *International Journal of Innovative Research in Computer and Communication Engineering (An ISO 3297: 2007 Certified Organization)* *Vol. 2, Issue 1,* January 2014.
4. Jubair Al-Ja’afer and Khair Eddin Sabri, ” Metrics for Object Oriented Design (MOOD) to Assess Java Programs,” in *International Arab Conference on Information Technology (ACIT’04)*, Algeria, 2004.
5. Dr Sonal Chawla and Gagandeep Kaur, “Comparative Study of the Software Metrics for the complexity and Maintainability of Software Development”, *(IJACSA) International Journal of Advanced Computer Science and Applications*, Vol. 4, No. 9, 2013.
6. Jeenam Chawla and Arun Agarwal, “Object-Oriented Design Metrics to Predict Fault Proneness of Software Applications,” *(IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 5 (3)* , 2014, 3318-3321*.*
7. U. L. Kulkarni, Y. R. Kalshetty, and Vrushali G. Arde, “Validation of CK metrics for Object Oriented Design Measurement,” Third International Conference on Emerging Trends in Engineering and Technology 978-0-7695-4246-1/10 $26.00 © 2010 IEEE DOI 10.1109/ICETET.2010.159.
8. T. Gyimóthy, Rudolf Ferenc, and Istva´n Siket, “Empirical validation of object-oriented metrics on open source software for fault prediction,” in [*IEEE Transactions on Software Engineering*](http://ieeexplore.ieee.org/xpl/RecentIssue.jsp?punumber=32) (Volume: 31, [Issue: 10](http://ieeexplore.ieee.org/xpl/tocresult.jsp?isnumber=32934), Oct. 2005).
9. S. Dubey and A. Rana, "Assessment of maintainability metrics for object-oriented software system," in *ACM SIGSOFT Software Engineering Notes, vol. 36, no. 5, p. 1,* 2011.
10. Programmer's Friend Class Dependency Analyzer (CDA) V1.16.0 [Online]. Available: <http://www.dependency-analyzer.org/>
11. Dependency Analysis with STAN (Video) [Online]. Available: <http://stan4j.com/>
12. Java Dependency Analysis Tool [Online]. Available: https://wiki.openjdk.java. net/display/JDK8/Java+Dependency+Analysis+Tool
13. An Introduction to Object-Oriented Metrics [Online]. Available: http://agile.csc.ncsu.edu / SEMaterials/OOMetrics.htm