Metric Analysis

Mani Rathnam Kasoju

Hema Bharath Kumar Narra

Section 1

● Introduction

This report's primary goal is to study C&K Metrics, which are useful for evaluating software quality attributes. We are looking at five distinct GitHub projects with the C&K tool to concentrate on the testability, maintainability, and reuse of software components. The results of the analysis of the five various projects are presented in this paper. The importance of C&K metrics in assessing software quality traits is discussed in the report's conclusion.

● GQM 1

Objective*:* This project's goal is to identify the flaws that affect the software's quality.

Question: What degree of fault-finding effectiveness does the C&K tool have?

Metrics: To determine code metrics like class degree and method degree in Java projects, the CK tool provides a wide range of metrics and makes use of static analysis techniques. We will concentrate on using the CBO (Coupling Between Objects), LOC (Lines of Code), RFC (Response for a Class), and WMC (Weighted Methods per Class) metrics for this project. These measurements will be essential in locating errors and defects in the examined codebase.

● GQM 2

Objective: To determine whether code reuse is possible.

Question: Whether the projects encourage the potential for reuse?

Metrics: We can assess the possibility of reusability by considering variables like connectivity between objects, lack of method cohesion, depth of inheritance tree, and weighted methods per class.

Section 2

I have selected five projects from GitHub for my project, and one of them is Maven SCM. Maven SCM is a software project management and comprehension tool that utilizes a project object model (POM) to control a project's build, reporting, and documentation from a single source of data. It is commonly used in Java projects and is known for its efficiency in managing software development projects (Antony, 2013)

Maven Wagon, the second Java project I've chosen, is essential to the Maven ecosystem since it offers the functionality required for connecting with repositories and transferring items. The deployment and retrieval of artifacts is made simpler by its transport abstraction layer, which also ensures stability and compatibility with different repository types.

The third Java project that I have chosen is Automator, a potent Android software that enables users to automate chores and build custom workflows by integrating various applications. Its objectives include increasing user productivity, lowering manual labor requirements, and giving Android device users a more effective and practical experience.

The fourth Java project I've chosen is Lunar Gdx, a useful networking tool for programmers creating LibGDX-based multiplayer game applications. It makes networking easier and offers necessary utilities, allowing developers to concentrate on making interesting and dynamic multiplayer game experiences.

The Terasology behavior system, along with its voxel-based world and modular design, offers a solid platform for game creation, making it the fifth Java project I've chosen. It lets programmers make dynamic, customizable environments that are immersive and participatory. As a result of the project's collaborative character, a wide variety of abilities and knowledge are guaranteed, encouraging innovation and originality in the creation of gameplay scenarios in voxel environments.

The datasets from the various initiatives can now be compared. Many metrics, including classes, CBO, LOC, RFC, and WMC, are accessible in the dataset. The attributes of the dataset are shown in the table below.

Project name	**Total CBO**	**Total LOC**	**Total RFC**	**Total WMC**
Maven-SCM	3	4.5	2.5	1.5
Maven-wagon	6	14	4	2.5
Automator	4.5	14.5	6.5	4
Lunar Gdx	2.5	22	4.5	7
Terasology Behaviors	4.8	25	3.5	2.8

Table 1: characteristics of the datasets

Several projects, including Maven-SCM, Maven wagon, Automator, LunarGdx, and Teratology behavior are covered in the table. For each project, it displays the overall number of classes as well as the CBO, LOC, RFC, and WMC. With 430 classes, the Maven-SCM project has the most classes overall. It also has the highest values for CBO, LOC, RFC, and WMC.

Section 3:

The CK code metrics were used in this study to analyze the class structure, message passing, inheritance, and encapsulation of Object-Oriented systems. The complexity and properties of Object-Oriented systems can be evaluated using a variety of metrics provided by CK metrics.

a) **Coupling between Objects (CBO**): This term refers to the dependencies between classes and can be calculated by counting the number of dependencies a class has. A tool that scans a class can determine the types that are used there, revealing information about how well-coupled the class is to other classes.

b) **Lines of Code (LOC):** LOC is a statistic used to gauge the size of a program's source code. It gives an indication of the size and complexity of the software. The internal representation of the source code is used by the LOC measure to precisely calculate the count while excluding comments and empty lines.

c) **Response for a Class (RFC):** RFC counts the number of different method calls that a class is accountable for to determine its level of responsibility. It gives a hint as to the role the class played in the general system behavior. When methods have numerous overloads with the same number of parameters but different types, RFC may be constrained.

d) **Weighted Method Count (WMC)**: WMC is a metric that assesses a class's complexity by counting how many branches there are in its methods. It gives information about the class's complexity by reflecting the variety of different execution pathways within the class.

g) **Fan-In**: Fan-In counts the classes that depend on or refer to a specific class. The extent of the interconnection of the class inside the software system is indicated, and the incoming dependencies are quantified. Fan-In takes object instances, method calls, and attribute references into account.

f) **Tight Class Cohesion (TCC)**: TCC measures how tightly two methods or the invocation trees of those methods overlap when accessing the same class variables. It delivers a number between 0 and 1, with higher numbers denoting greater coherence and lower numbers denoting less cohesion. The metrics CBO, LOC, RFC, and WMC were picked for this report's analysis of the chosen Object-Oriented systems. According to Chidamber, these measures offer important insights into the characteristics and complexity of the systems under study.

Section 4:

The results of applying the Ck code measure to GitHub's five various Java projects are shown in the charts below.

The first project, Maven-SCM, yielded the results shown below. The ck metrics tool's chart displays the results.

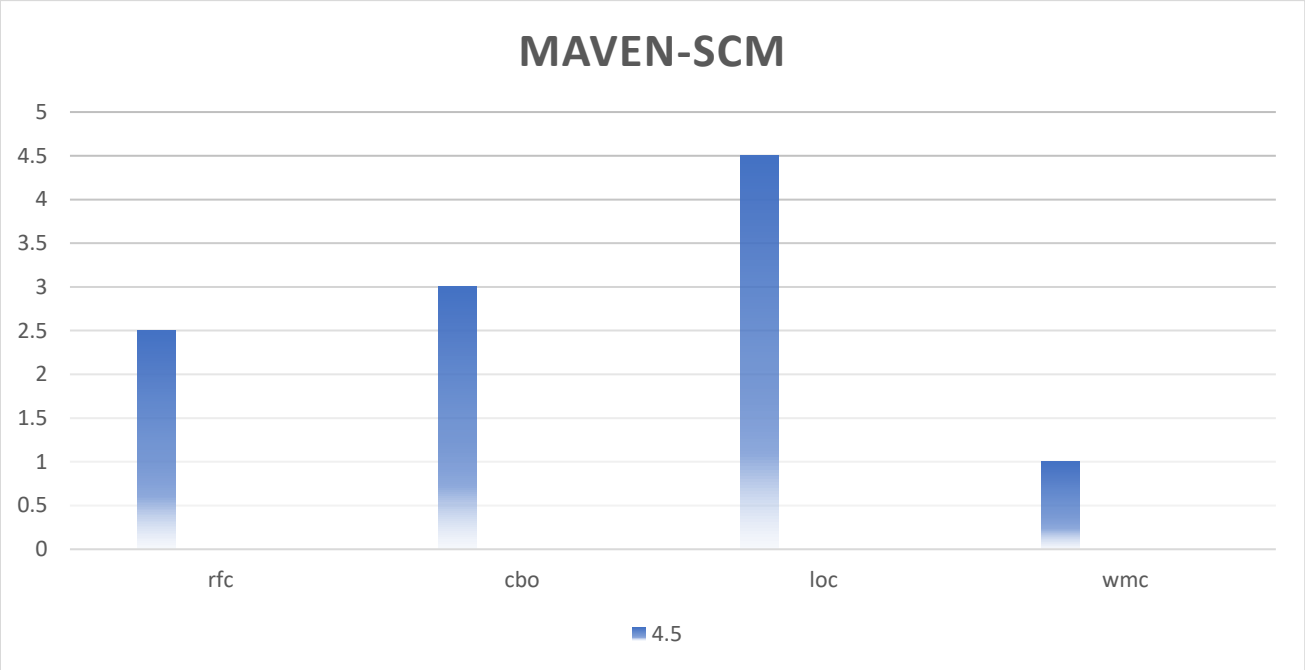

Fig 1: a graphic representation of the outcomes of the Maven-SCM project

With the help of the chart in Fig., the outcome of the project, which is the Maven-wagon, is displayed in detail.

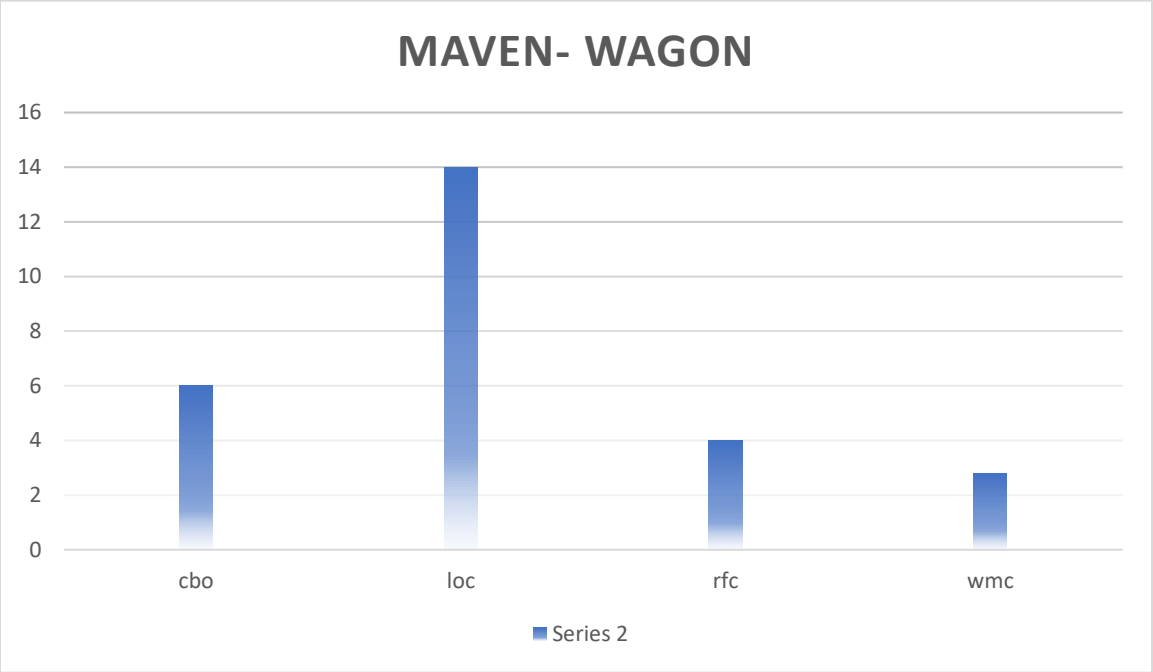

Fig 2: outcomes for Maven-wagon

The results for project three, the Automator, are shown in more detail in the following Fig 3.

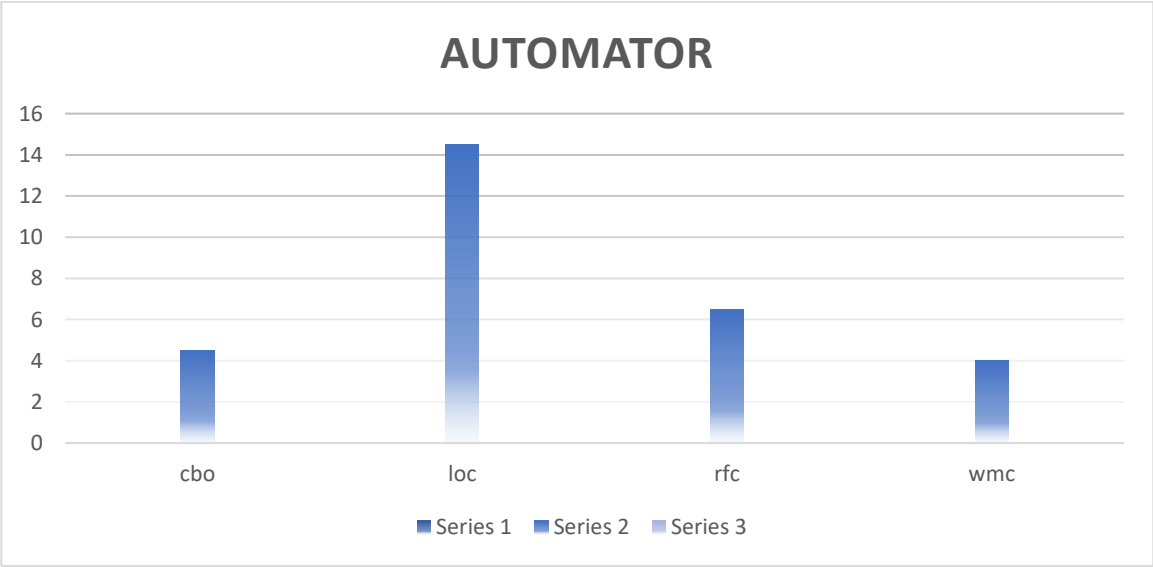

Fig 3: the project Automator's results.

The following outcome is given for the fourth project, Lunar Gdx. The details of the result are displayed in Fig. 4. The results of this project are shown in the chart.

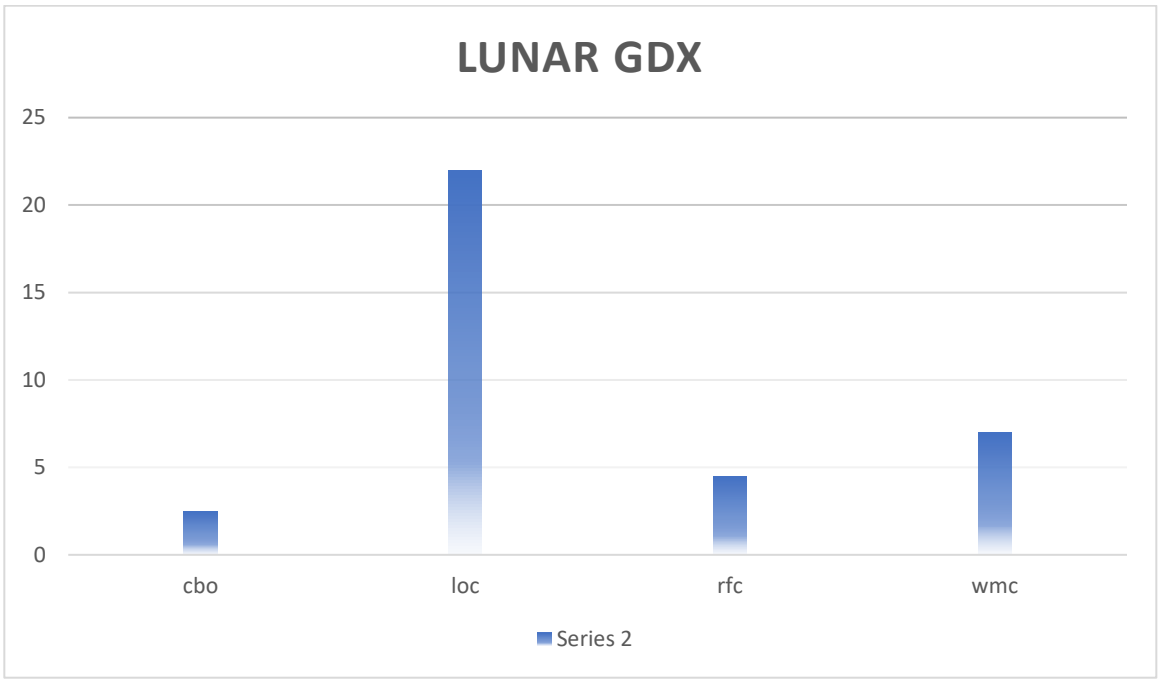

Fig 4: outcomes for project Lunar Gdx

The following result is given for project five, which is titled Terasology Behaviors. The outcome is displayed in Fig. 5 below.

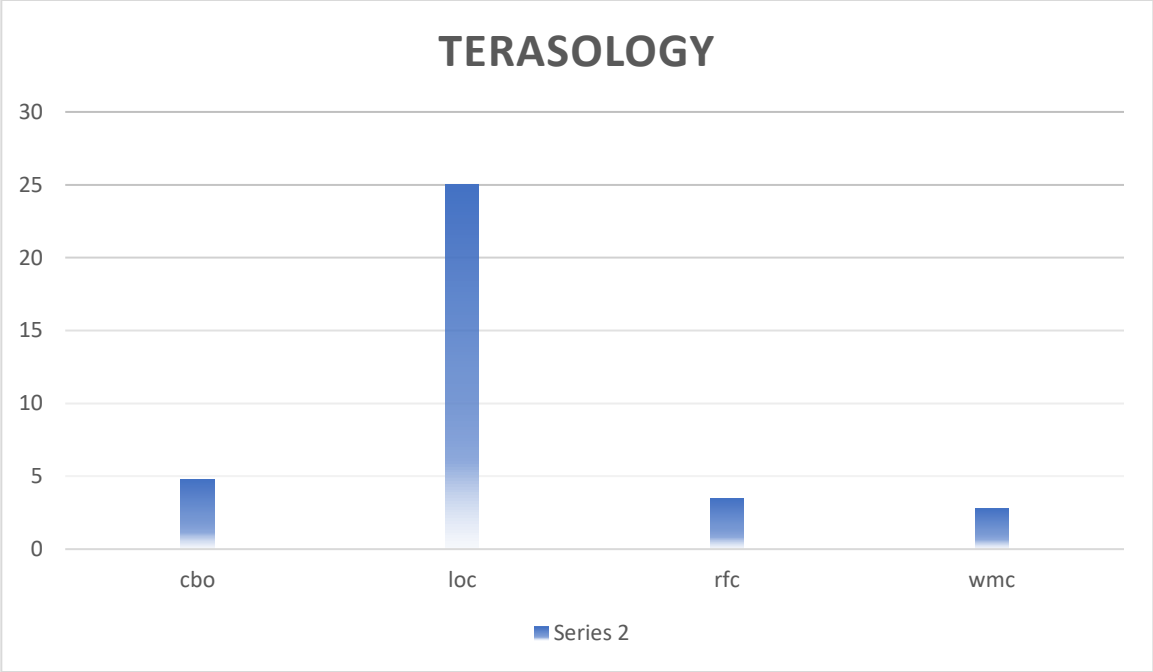

Fig 5: outcome for terasology projects

Section 5

Conclusion:

The software quality attributes of five Java projects that were downloaded from GitHub were assessed in this report using CK code metrics. Measurement of the software's testability, reusability, and maintainability properties was the analysis's main objective. Terasology, Maven SCM, Lunar Gdx, Automator, and Maven Wagon were the projects chosen for this investigation.

The complexity and features of classes in these projects were assessed using the CK criteria. Using methods, classes define an object's structure and behavior, acting as a blueprint for object creation. The metrics that were selected—CBO, LOC, RFC, and WMC—offered insights into various facets of class complexity and dependencies.

The number of dependencies that a class has was quantified using CBO, which symbolizes the coupling between classes. By calculating the number of lines of code, LOC assisted in

determining the length and complexity of the programs. RFC calculated how many unique method invocations a class is accountable for, giving information on its duties. WMC calculated the difficulty of a class by counting the number of branches within it.

We received results that provided insight into the software quality characteristics of the chosen projects by applying these metrics. The research gave important insights into the projects' testability, reusability, and maintainability. These metrics are useful for evaluating the codebase and pinpointing places where software quality needs to be improved.

Overall, this assignment's usage of CK code metrics helped assess the software quality of the chosen projects and gave students a thorough understanding of their code bases. The results of this study can be utilized to direct future work and raise the projects' general level of quality.

Reference:

Löwe, W., Lincke, R., & Lundberg (2008, July). software metrics tool comparison. 2008 International Symposium on Software Testing and Analysis Proceedings, pp. 131–142.

U. L. Kulkarni, Y. R. Kalshetty, & V. G. Arde (2010). Validation of ck metrics for measuring object-oriented design. The third international conference on cutting-edge engineering and technology was held in 2010 (pp. 646–651). IEEE.

J. P. Antony (2013). software reliability prediction using CK metric thresholds. Global Advanced Networking and Applications Journal, 4(6), 1778.

(2003). Subramanyam, R., and Krishnan, M. S. Software faults are affected by the empirical examination of CK metrics for object-oriented design complexity. 29(4), 297-310, IEEE Transactions on Software Engineering.

S. R. Chidamber, D. P. Darcy, and C. F. Kemerer (1998). Utilizing metrics in management for object-oriented.