# OBJECT ORIENTED DEVELOPMENT

**Group-4 Assignment-1**

**Empirical Study: Effect of Class size on software maintainability**

**Group members:**

**1: Durga Vijaya Ramaraju Pamidi**

1. **Nithin Esam**
2. **Priyanka Pulikanthi**
3. **Chanikya Reddy Guntakrindapalli**

**Github link:**

https://github.com/Durga-Vijaya-Ramaraju/Object-Oriented

Section 1:

The Goal-Question-Metric (GQM) approach has been an established method to be applied in academic and industrial research. It had been developed originally for NASA projects at the Goddard Space Flight Center. Based on the basic idea that for measurements to make sense, the study must first clearly define its goals. These goals are linked to specific questions, which interpret them, and finally, metrics are identified to measure these goals and answer the questions.

Goals define what we will be trying to accomplish.

Questions help evaluate if these goals are met and provide context to interpret the findings.

Metrics are the measurable indicators needed to address the questions and assess progress toward the goals.

From the Assignment1 view, the goal is to conduct an empirical study: Effect of class size on software maintainability.

Question: what is the effect of the class size on software maintainability?

Metrics:

Introduced by Chidamber and Kemerer, best know as (C&K) metrics suite. Designed to:

Measure aspects of OO software

Measure the complexity of the design

Improve software development.

**CBO (Coupling Between Objects)**

**RFC (Response for Class)**

**Lines of Cod**

Section 2:

Subject Programs

|  |  |
| --- | --- |
| **SNo** | **Project Name** |
| 1 | Computer Store |
| 2 | JUnit |
| 3 | Design patterns Java Main |
| 4 | Hospital Management System |
| 5 | Petclinic-main |

Section3:

CK metrics, also known as Chidamber and Kemerer metrics, are a set of software metrics used to assess the complexity and maintainability of object-oriented software systems. They were introduced by Shyam R. Chidamber and Chris F. Kemerer in their paper "A Metrics Suite for Object-Oriented Design" in 1994.

The CK metrics aim to measure various aspects of software quality and provide insights into the design and maintainability of object-oriented systems. These metrics are computed by analyzing the structure and relationships among the classes in the system.

Here are some commonly used CK metrics:

Metrics

Weighted Methods per Class (WMC): Quantifies class complexity. Higher values indicate a difficulty in understanding and modifying the class.the metric measures the complexity of a class by summing up the totalnumber of methods. A higher valuefor WMC indicates greater complexity in the class.

* Depth of Inheritance Tree (DIT): Stores the depth of class hierarchy. I decides dependencies and potential maintenance costs. in terms of the number of levels between a class and its root inheritance hierarchy. The higher the value of DIT, the deeper the inheritance structure is; hence, the higher the complexity.
* Number of Children (NOC): The number of direct subclasses of a class measures how specialized it is. The higher the value of NOC, the more specialized a class is, and likely the more complex the system becomes.
* Response for a Class (RFC): Number of methods called. High RFC means class interactions and complexity are increasing.
* Coupling Between Objects (CBO): This metric measures the number of other classes that a class is directly dependent on. It indicates interdependence among classes. Thus, the higher the value of CBO, the tighter the coupling; the lesser the coupling, the better for system maintenance and understandability.
* LCOM: Measures the extent of lack of cohesion in methods. It calculates how well the methods in a class are related to one another, based on the instance variables they use. A higher value of LCOM means lower cohesion; it may indicate poor design and possible problems in future maintenance of the class.

JUnit

GitHub Repository: ( <https://github.com/junit-team/junit5.git> )

JUnit is an open-source, Java-based testing framework designed for fast creation and execution of tests at any stage in the development life cycle. Its modular architecture introduced in JUnit 5 supports various sizes of projects and promotes better stability of code with rigorous testing practices. JUnit is especially important for TDD, where writing tests comes before writing code, ensuring that the software behaves as expected.

The presence of features such as test runners, fixtures, and assertions makes unit testing, integration testing, and regression testing easier with JUnit. Its minimum DIT score is 1.68, which means little hierarchical dependencies; hence, the codebase is less complex and thus easier to manage. Furthermore, its WMC of 11.28 indicates a somewhat higher complexity level, and a CBO score of 6.19 shows relatively low inter-class dependencies, which makes the classes much easier to maintain.

In general, the simplicity and thoroughness in testing emphasized by JUnit significantly enhance software quality; it is an important resource for Java developers working in small- and large-scale environments.

Spring PetClinic

GitHub Repository: <https://github.com/spring-projects/spring-petclinic.git>

Spring PetClinic is an open-source Java application that demonstrates the capabilities of the Spring Framework in the real-world example of a veterinary hospital management system. The project has been developed on the principles of modular, layered architecture using Spring Boot, Spring MVC, and Spring Data JPA technologies. Those technologies allow fast application configuration and efficient database management with little boilerplate code. The Weighted Methods per Class of 3.67 is relatively lower, hence less complex it is. Coupling Between Objects: 4.17, fewer interdependencies, making it modular, hence more maintainable. Moreover, the Depth of Inheritance Tree has a value of 1.18, which provides shallow inheritance, meaning very few, and that makes it more desirably clear in terms of the class structure for better manageability of code. Spring PetClinic is a good starting point for developers getting into the Spring ecosystem, showing best practices on how to implement and integrate Spring technologies while keeping complexity at bay.

Section4:

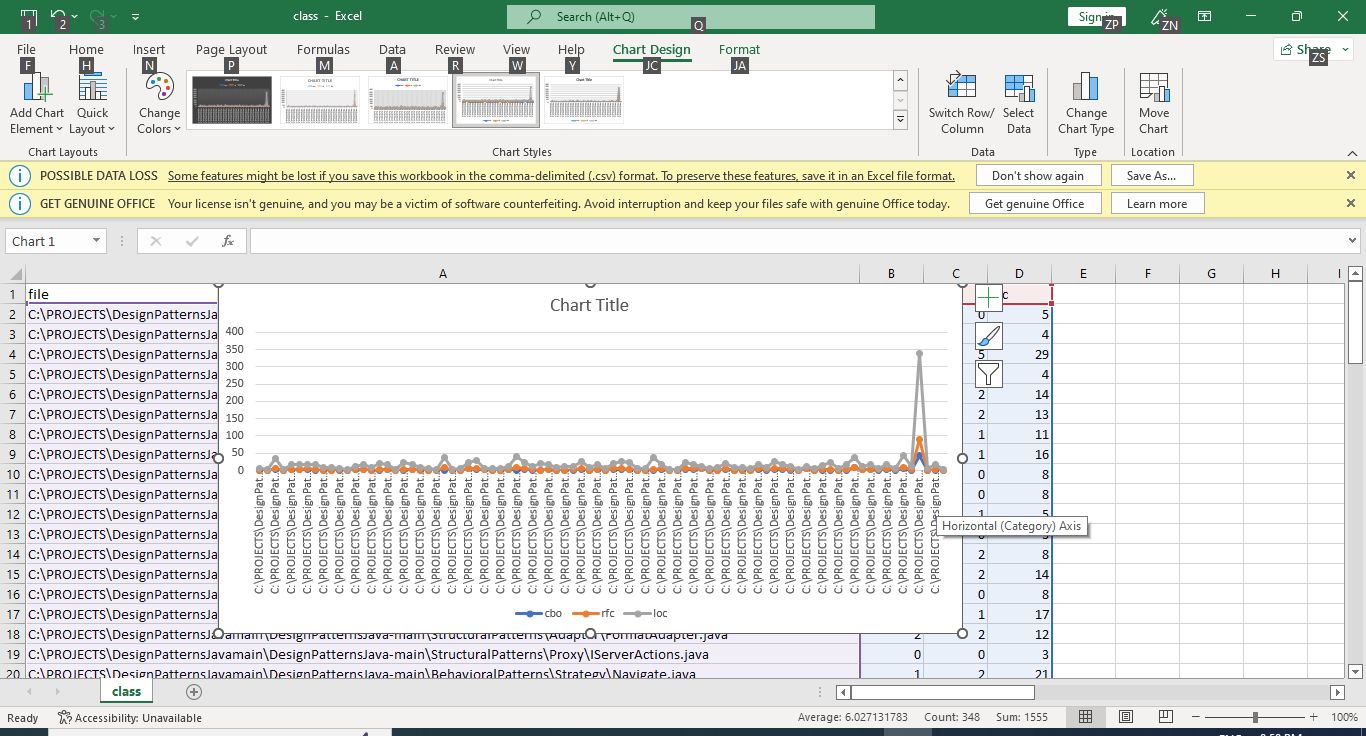
|  |  |  |  |
| --- | --- | --- | --- |
| Project Number | CBO | RFC | LOC |
| Computer Store | 1.74 | 1.97 | 10.55 |
| JUnit | 11.28 | 6.19 | 1.68 |
| Design patterns | 2.15 | 3.92 | 16.07 |
| Hospital Management Sys. | 2.12 | 1.79 | 14.16 |
| Pet-clinic main | 3.67 | 4.17 | 1.18 |

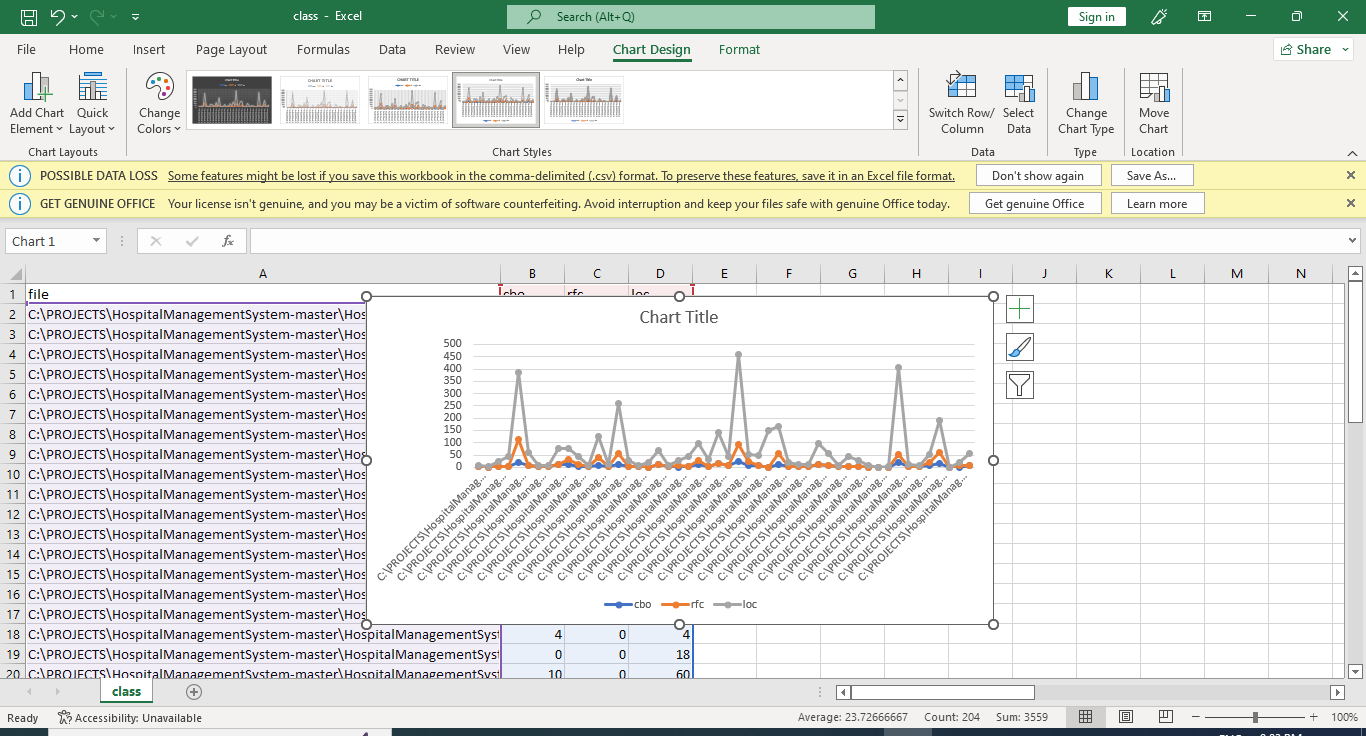
Higher WMC means that each class has an average moderately high level of method complexity, which may impact maintainability for large codebases.

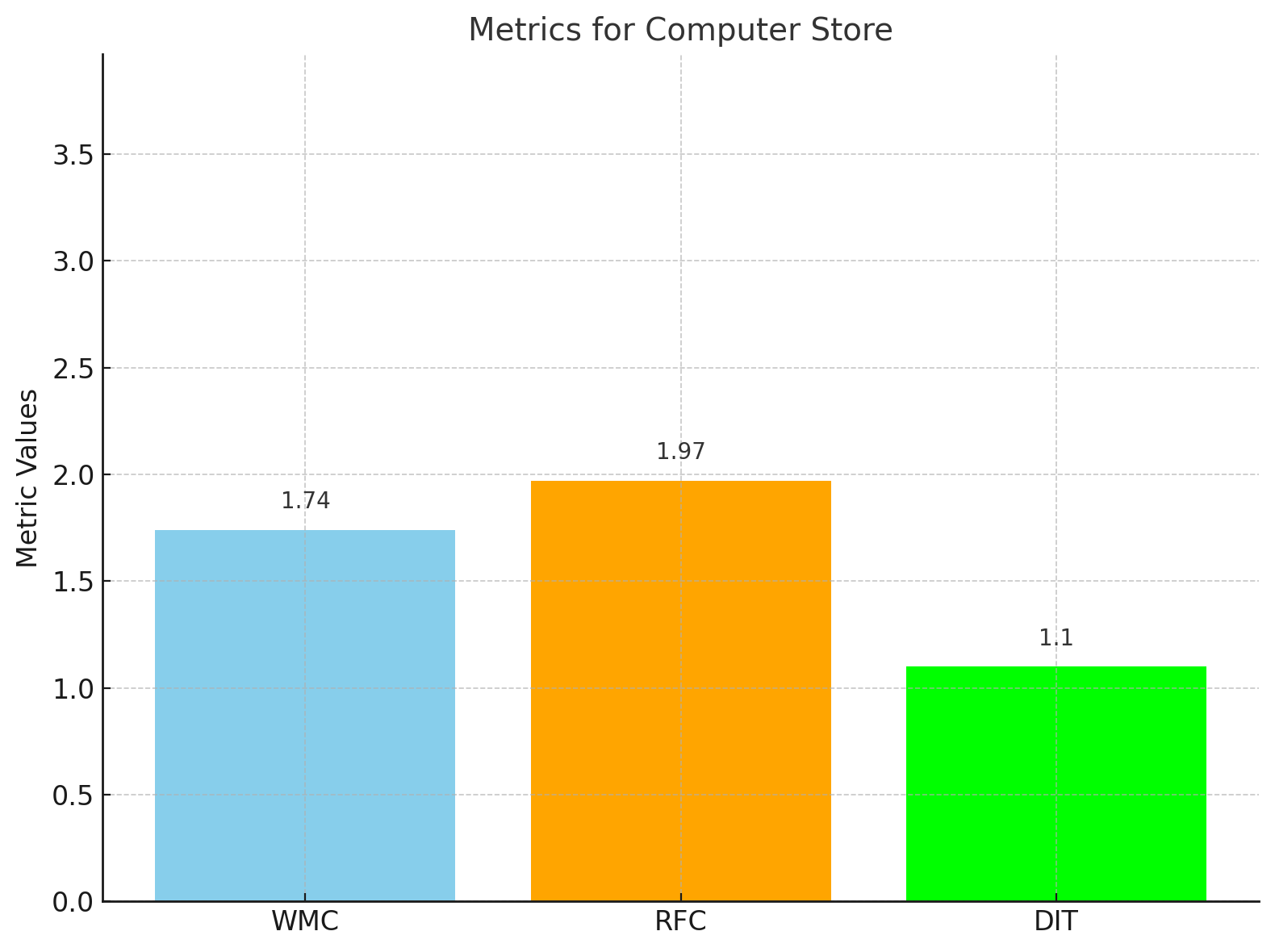
Low CBO and DIT scores indicate fewer interdependencies and a shallow inheritance hierarchy, which increase modularity and stability of the code.

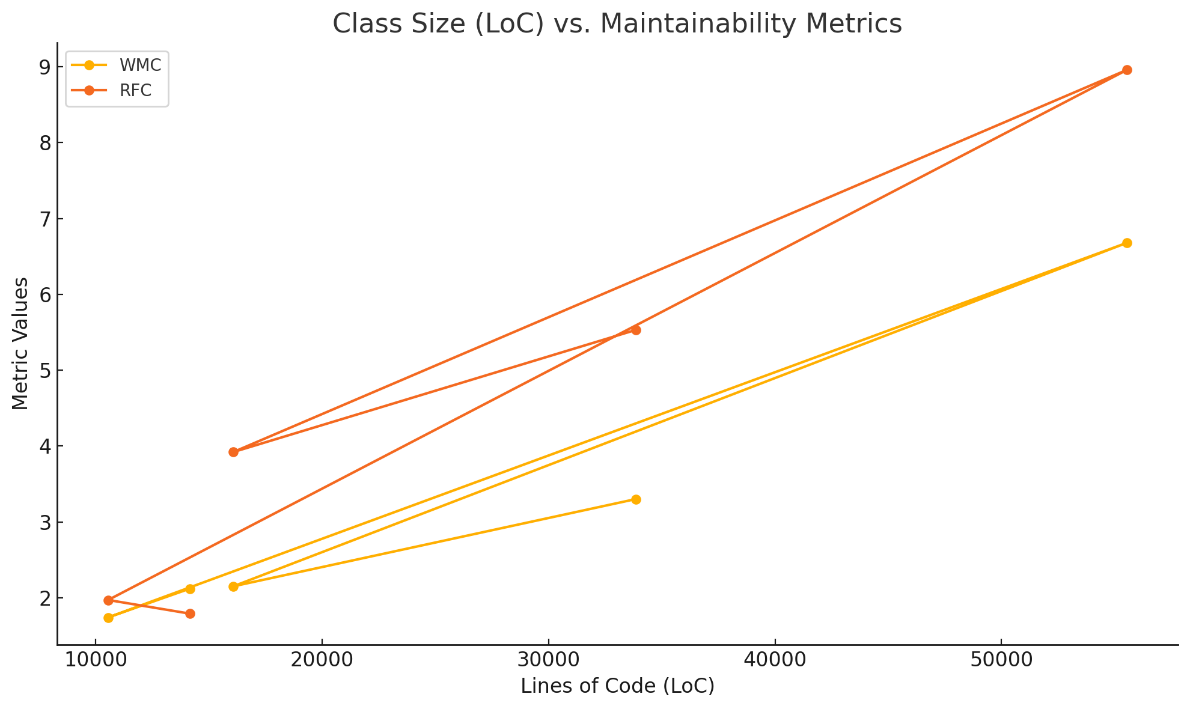
Lower WMC means less complex methods per class, which encourages easier understanding and modification.

CBO and DIT values indicate that this is well-structured with less interdependency and shallow inheritance, which provides better maintainability.









Overview:

The complexity of the observed courses is moderate, as shown by the average WMC. This may impact maintainability since higher complexity usually makes classes harder to understand and change.

Insights drawn:

The classes studied show a moderate degree of method interactions, as reflected by an average value of RFC. That is evidence of a balanced volume of incoming and outgoing messages, which might increase complexity and additional maintenance challenges.

Average values for WMC and RFC indicate that classes under study can be subjected to maintainability problems due to their complexity and method interaction. Higher values of these metrics usually flag lower maintainability, as they denote higher complexity and possible problems when modifying classes.

Although no detailed values of DIT are presented, inheritance hierarchies can still impact maintainability. When there is a multiple-level inheritance structure, more dependencies and added complexity are likely to emerge, making maintenance of observed classes even harder.

Conclusion

The data suggests that complexity, as measured by WMC and RFC, has an impact on maintainability. Maintenance activities might be planned with care, since the system is moderately complex and methods interact with each other; more importantly, inheritance hierarchies could add more obstacles to maintainability. More in-depth investigation of this relation is possible by analyzing a larger dataset.

A low Coupling Between Objects (CBO) value enhances modularity, which supports encapsulation and improves the reusability, maintainability, and testability of the code. Conversely, a high Response for a Class (RFC) value indicates increased complexity, making the class more challenging to test and maintain.

In this project, we have mainly analyzed three metrics, which are CBO, RFC, and LOC. The selected projects are showing the low value of these metrics, hence class size does affect maintainability. Low values of CBO will then indicate better modularity, which increases encapsulation, reusability, maintainability, and testability. On the other hand, high values of RFC increase complexity, making the class difficult to test and maintain.

References:

<https://www.geeksforgeeks.org/goal-question-metric-approach-in-software-quality/>

R. Subramanyam and M. S. Krishnan, "Empirical analysis of CK metrics for object-oriented design complexity: implications for software defects," in *IEEE Transactions on Software Engineering*, vol. 29, no. 4, pp. 297-310, April 2003, doi: 10.1109/TSE.2003.1191795

<https://www.aivosto.com/project/help/pm-oo-ck.html>