Rajesh Rachamalla

Assignment 1

Object-Oriented Development

Lewis University

Instructor Fadi Wedyan

Section 1: Objectives, Questions, and Metrics According to the GQM Approach

Objective

This empirical research aims to examine how the class size of software affects the maintenance of the chosen open-source Java web applications, that is, OpenPDF, JUnit4, Karate, Log4j 2, and OpenBoard. The size of the classes in Lines of Code (LoC) is often linked with the more complex structure and larger scope of responsibility. Classes can be functional and can accumulate a functional capability as they grow larger, creating an increase in coupling and internal complexity that may have a negative effect on maintainability. This paper uses the Goal-Question-Metric (GQM) paradigm to critically assess whether bigger classes are characterized by worse maintainability. Maintainability is measured based on chosen Chidamber and Kemerer (C&K) measures, namely Coupling Between Object Classes (CBO) and Weighted Methods per Class (WMC), to empirically find out the structural effect of size.

Goal

This paper aims to discuss the size of classes in some open-source Java projects as a measure of the effect of class size on the sustainability of software. In particular, the paper discusses the issue of Lines of Code (LoC) increase being linked to increased complexity and coupling, in the form of WMC and CBO. The hypothesis that is to be tested in the analysis is that larger classes are less maintainable than small, more modular classes.

Questions

1. To what extent does the class size (in terms of LoC) affect maintainability measures of the chosen Java projects?
2. Does it show that larger classes are more likely to be more coupled (CBO)?
3. Are bigger classes more internally complex, as shown by higher WMC values?
4. Are there any large classes that may be considered as possible maintainability risk points in the considered systems?

Metrics

In accordance with the Goal-Question-Metric (GQM) paradigm, the metrics were chosen to respond to the research questions about the effect of the size of the classes on maintainability. The independent variable in the research is the class size that will be measured in terms of Lines of Code (LoC) in the class. The maintainability is measured with two Chidamber and Kemerer

(C&K) measures, namely Coupling Between Object Classes (CBO) and Weighted Methods per Class (WMC) (Jangra, et al., 2022).

CBO measures the external class dependencies, which represent structural coupling and change impact. Greater CBO implies lower maintainability as a result of greater inter-class dependency. WMC is a measure of the cumulative complexity of methods in a class, which is an internal structural complexity. Increased WMC implies increased maintenance. These metrics, in combination, operationalize maintainability in structural ways in agreement with the purpose of the study.

Section 2: Description of the Subject Programs (Dataset)

In this empirical study, I chose 5 open-source Java projects on GitHub.The criteria used in the selection were:

- At least 10000 lines of code
- History of active development (more than one)
- The project is 3+ years, and it ensures natural evolution and modularity degradation.
- Code to be analyzed: Open Source code.
- Can be defined and studied by using static analysis.

These standards guarantee significant modularity testing over developed software systems.

Name of Program	Description	Number of Classes	Avg CBO	Avg LCOM
OpenPDF	A Java library to create and manipulate PDF documents, which is popular with enterprise reporting systems.	1745	5.93	91.47
JUnit4	An automated Java testing framework, and assertion, and test runners are used.	1,471	3.86	12.96
Karate	A test automation system based on BDD, which can be used to undertake API testing, performance testing, and service validation.	616	6.60	113.61

Log4j2	Apache enterprise logging platform that provides asynchronous, configurable, and performant logging pipelines.	3,432	4.50	46.23
OpenBoard	A classroom and presentation interactive digital whiteboard that is open source.	414	5.96	47.02

This empirical research examines 5 open-source Java systems that are mature: Open PDF, JUnit4, Karate, Log4j2, and openBoard. The chosen projects meet the predetermined inclusion criteria: (1) large size of codebase, (2) history of at least three years of development, (3) current maintenance, and (4) appropriateness to be analyzed with the CK metrics tool (Macia, et al., 2011). These conditions make sure that the systems are realistic when developing software products, which have been maintained, refactored, and expanded through the years. Such systems can be studied, and the relationship between maintainability and class size can be safely observed.

OpenPDF is a Java library for creating and manipulating PDF documents in enterprise and reporting systems. Since it has document processing capabilities and backward compatibility, it has classes of different sizes and responsibilities.

JUnit4 is a popular unit testing computing structure conduction of Java, which offers annotations, claims, and testing execution systems. It has a reputation of having a rather clean and modular design.

Karate is Behavior-Driven Development (BDD) that is mainly applied to API and service testing. It combines testing logic with scripting facilities, producing an easy-weight utility and heavy-integration classes blend.

Apache created a logging framework called Log4j2, which is used by enterprises. It also promotes pipelines that are configurable and asynchronous logging, which adds complexity to the inter-component architecture.

OpenBoard is an interactive whiteboard application that is used in educational settings. It is loaded with graphical interface logic and classroom-based functionality, resulting in different class designs.

These systems, in combination, offer diverse domains and architectural styles and allow the overall analysis of the effects of class size on maintainability.

Section 3: Description of the Tools Used

To conduct the empirical analysis, two static analysis tools were used: the CK Metrics Tool and PMD.

Tool 1: CK Metrics Tool

Class-level Chidamber and Kemerer (C&K) metrics of each of the selected Java projects were calculated using the CK Metrics Tool. The tool does a static analysis and produces CSV files with structure measurements like Lines of Code (LOC), Weighted Methods per Class (WMC), Coupling between Object Classes (CBO), RFC, DIT, NOC, and LCOM (Kulkarni, et al., 2010). In this research, class size was measured by using LOC, and maintainability was measured using WMC and CBO. The tool is able to extract metrics that are both automated, consistent, and reproducible on large codebases (Macia, et al., 2011).

Command used

The tool was used through the command-line interface that is offered in its documentation, and results were exported per chosen project. The objective, repeatable, and non-intrusive measurement of structural properties ascertained by the method of static analysis is appropriate to empirical research of maintainability. Here is the command:

```
java -jar ck-x.x.x-SNAPSHOT-jar-with-dependencies.jar <project dir> true 0 true <output dir>
```

Tool 2: PMD Static Code Analyzer

PMD was employed to supplement the quantitative metric analysis to identify design imperfections in the structure and possible maintainability problems. PMD determines complicated classes, long methods, over-dependencies, and design principle breaches (Singh, et al., 2017). It has a rule-based analysis to offer qualitative information on the quality of the code and structural vulnerability that could be associated with large classes.

Command Used

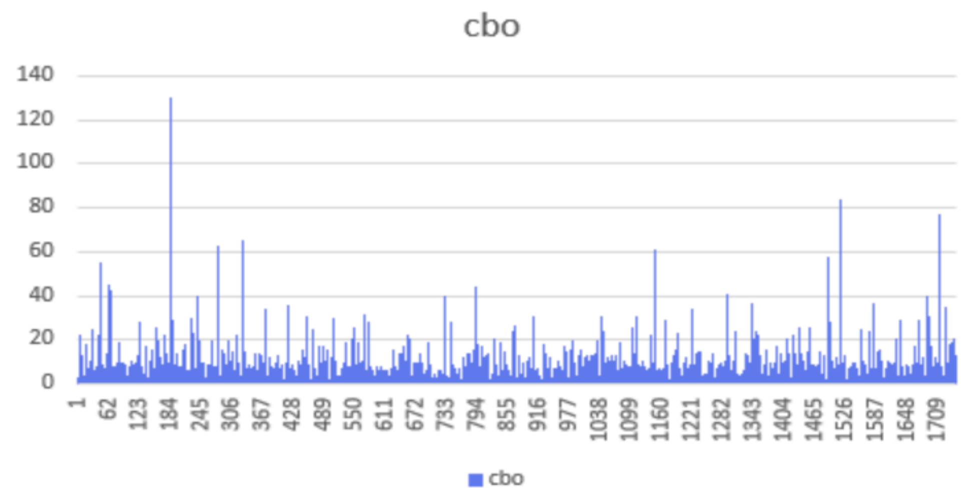
The tool was run on a command line with programmed rule-sets, and the generated reports were processed to assist in the interpretation of maintainability trends on the metric result outputs. Here is the command:

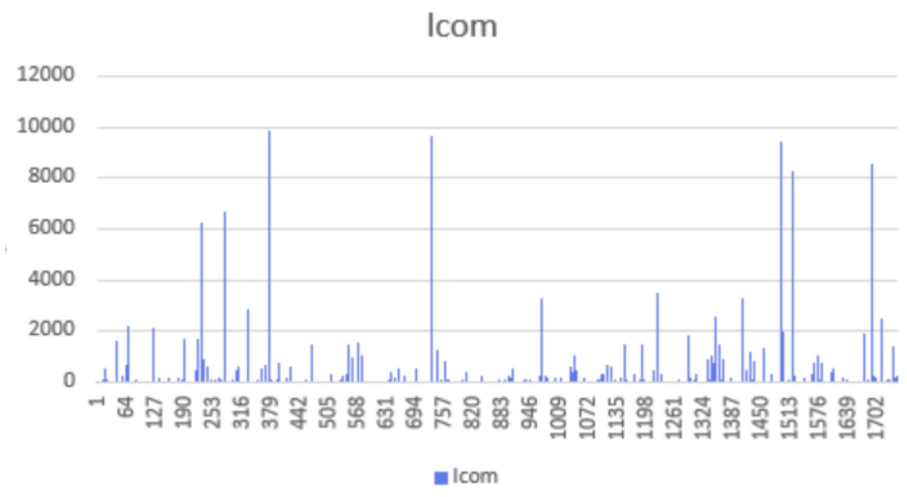
```
pmd.bat check -d <projectDir> -f xml -R ruleset.xml -r output.xml
```

Section 4: Results

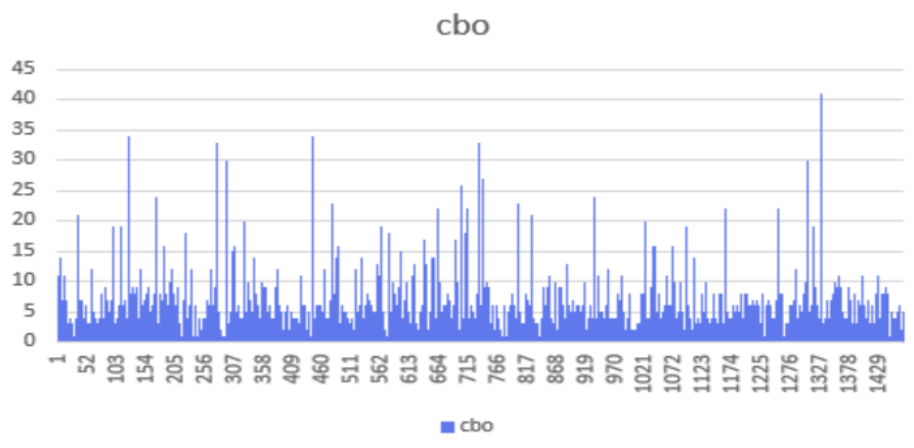
This section presents results from analyzing CBO and LCOM distributions across all projects.

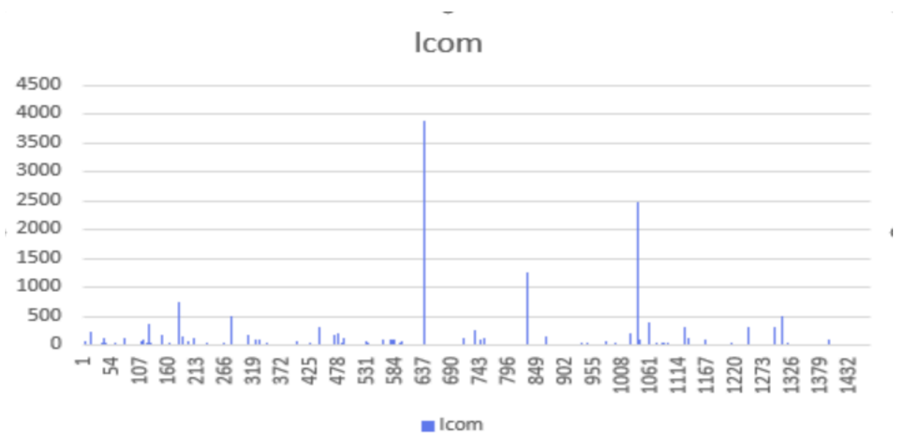
Project 1: openpdf-master



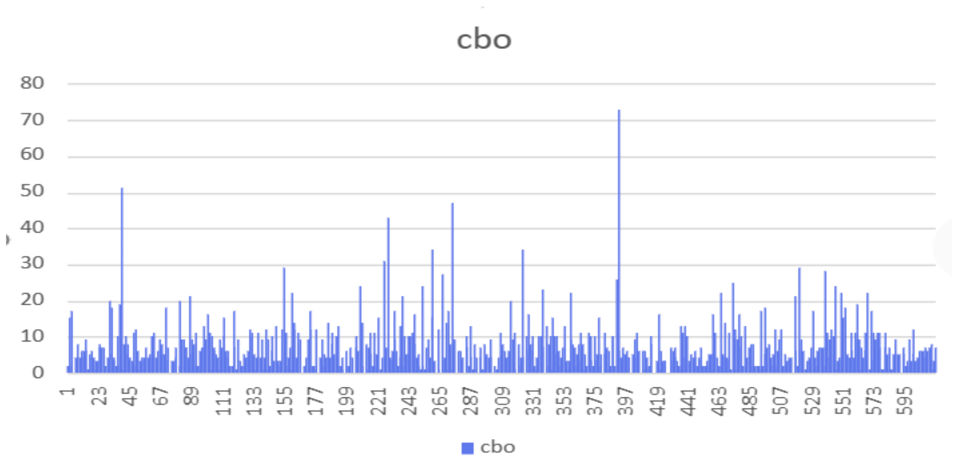


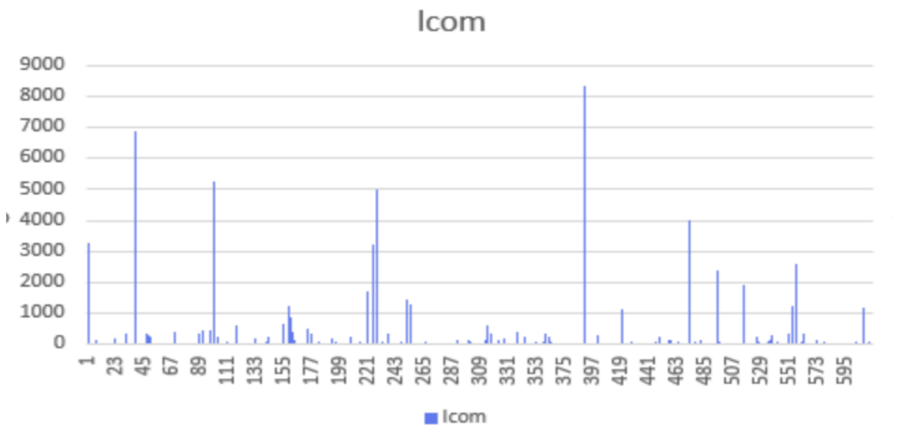
Project 2: junit4



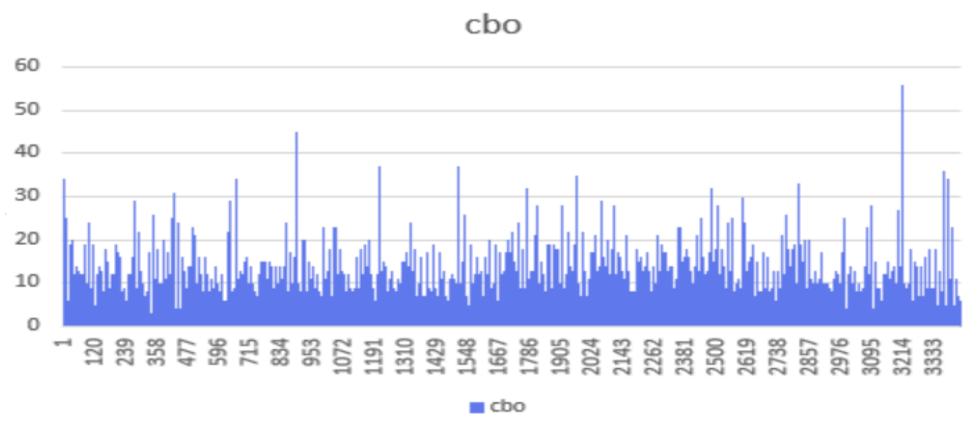


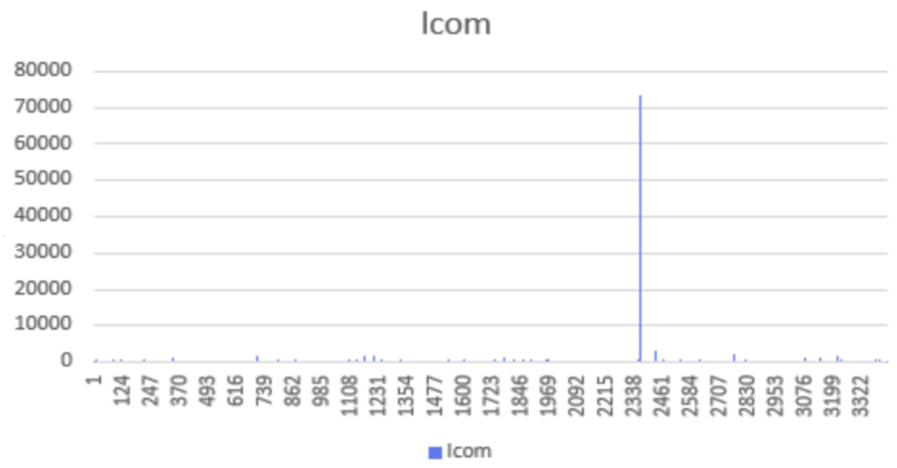
Project 3: karate-metric



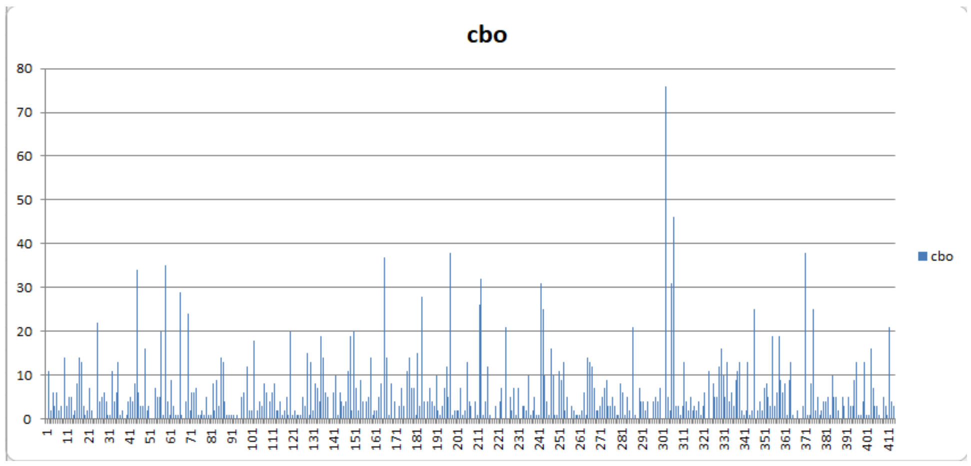


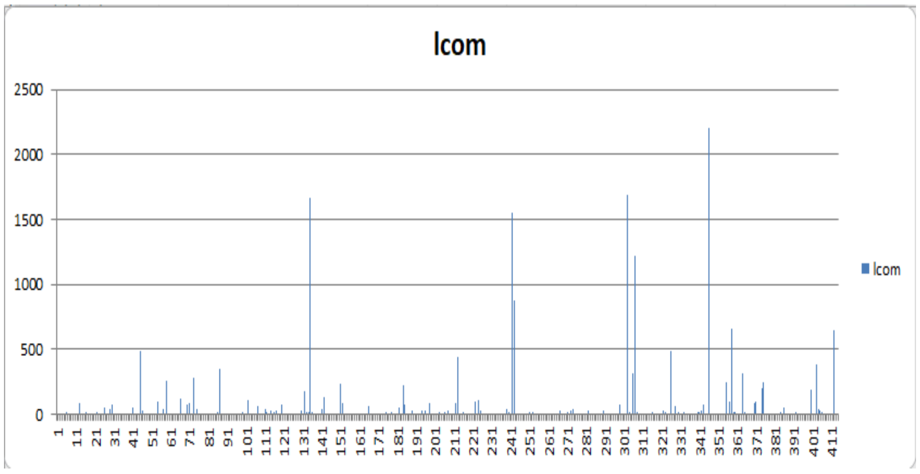
Project 4: logging-log4j2-2.x





Project 5: openboard





Observed Trends

Across OpenPDF, JUnit4, Karate, Log4j2, and OpenBoard, a consistent relationship was observed between class size (LoC) and maintainability metrics (WMC and CBO). The graphical and tabular results indicate the following patterns:

- ➢ Larger classes generally exhibit higher WMC, reflecting increased internal complexity.
- ➢ Increased class size is frequently associated with higher CBO, indicating stronger inter-class dependencies.
- ➢ Extremely large classes appear as metric outliers and may represent structural risk points.
- ➢ Projects with better size control (e.g., JUnit4) demonstrate more uniform metric distributions.
- ➢ Complex frameworks such as Log4j2 and OpenPDF show greater variation in size and coupling.

Overall, the data suggest that growth in class size contributes to structural complexity and reduced maintainability across the studied systems.

Section 5: Conclusion

It was an empirical study that analyzed how the class size influences software maintainability in 5 open-source Java projects, OpenPDF, JUnit4, Karate, Log4j2, and OpenBoard. The correlation analysis showed that the relationship between class size (LoC) and maintainability-related measures, namely, WMC and CBO, is positive and constant. The bigger classes tended to have more internal complexity and greater coupling, which boosts the maintenance effort and change impact. Projects with more managed class sizes, like JUnit4, were found to have more stable metric distributions and are better structurally balanced. On the contrary, projects with exceptionally big classes had localized hotspots of complexity. In general, the results confirm the hypothesis that high class size has an adverse impact on maintainability, and modular design, responsibility segregation, and constant refactoring are important practices.

Git link:- https://github.com/rajeshrachamalla-creator/OOD-Assignment1.git

Reference

GitHub Link:

Macia, I., Garcia, A., von Staa, A., Garcia, J., & Medvidovic, N. (2011, September). On the impact of aspect-oriented code smells on architecture modularity: An exploratory study. In *2011 Fifth Brazilian Symposium on Software Components, Architectures and Reuse* (pp. 41-50). IEEE.

Jangra, R., Sangwan, O. P., & Nandal, D. (2022, July). A Novel Approach for Software Effort Estimation using Optimized C&K Metrics. In *2022 Fifth International Conference on Computational Intelligence and Communication Technologies (CCICT)* (pp. 505-513). IEEE.

Singh, D., Sekar, V. R., Stolee, K. T., & Johnson, B. (2017, October). Evaluating how static analysis tools can reduce code review effort. In *2017 IEEE symposium on visual languages and human-centric computing (VL/HCC)* (pp. 101-105). IEEE.

Kulkarni, U. L., Kalshetty, Y. R., & Arde, V. G. (2010, November). Validation of ck metrics for object oriented design measurement. In *2010 3rd international conference on emerging trends in engineering and technology* (pp. 646-651). IEEE.