LEWIS UNIVERSITY

Object Oriented Development Group Assignment 2

By

Krishna Kiran Vinnakoata

Harish Reddy Moti

Gowtham Krishna Maganti

Date: 25 June 2023

Section 1

Objectives, questions, and metrics according to the GQM approach

Objectives:

- This empirical study aims to investigate the impact of code bad smells on modularity in Java projects.
- To define our metrics, we will adopt the Goal-Question-Metric (GQM) approach. Our goal is to assess the effect of code bad smells on the modularity of Java programs of varying sizes. We have formulated the following questions and metrics:

Goal: Empirical study on the effect of code bad smells on modularity.

Questions:

- What is the relationship between the bad smells of code and modularity in Java projects?
- How do different types of code bad smells influence the coupling and cohesion metrics (C&K metrics) in Java projects?
- What are the features of classes that have severe code smells and poor modularity?

Metrics: We will evaluate the selected metrics based on the following criteria for the subject programs:

- **Number of open issues > 0**: We will select programs with at least one open issue to ensure active maintenance and updates.
- **Size > 10000**: Programs of relatively large size will be chosen to provide sufficient code for analyzing bad smells and modularity.
- **Number of commits (>100):** Programs with a moderate number of commits will be selected to ensure an appropriate level of complexity without being too small or too large.

These criteria were carefully picked to ensure that the programs under consideration have a sufficient level of complexity and have experienced significant development, thereby generating useful data for our research. The size criterion is intended to research programs of modest scale, which are frequent in business. The number of commits requirement ensures that there is enough data for analysis.

Section 2

Describe the subject programs or what is also called Data set

For our study, we carefully handpicked ten Java projects from GitHub that align with our specified criteria. Table 1 provides a comprehensive overview of each program, featuring important attributes such as the project's name, description, size (in terms of lines of code), number of open issues, and total number of commits.

Program Name	Description	Size	Open Issues	Commits
Album-master	Album-master is project that aims to organize and manage a collection of albums.	69554	130	163
AndroidAll-master	AndroidAll is a project dedicated to providing a comprehensive collection of Android development resources and tools.	20049	3	411
DataX-master	It is a project that focuses on simplifying data integration and migration processes.	19714	947	350
GeometricWeather-master	Geometric Weather is a project that offers a comprehensive weather forecasting and information retrieval system.	986182	164	431
microservices-platform-master	Microservices-platform is a project that aims to simplify the development and deployment of microservices architectures.	250093	1	477
mybatis-generator-gui-master	MyBatis Generator GUI is a project that offers a user-friendly graphical interface for generating My Batis mapping and model files.	14954	107	267
react-native-date-picker-master	React Native Date Picker is a datetime picker for Android and iOS. It includes date, time and datetime picker modes. The date picker is customizable and supports different languages. It's written with native code to achieve the best possible look, feel and performance.	11367	29	488
sofa-ark-master	Sofa-ark is a project that provides a powerful and flexible runtime environment for running Java applications in a modular and isolated manner.	44673	43	258
SwipeRecyclerView-master	SwipeRecyclerView is a project that provides a customizable and feature-rich Recycler View implementation for Android applications	29347	106	108
testing-samples-main	A collection of samples demonstrating different frameworks and techniques for automated testing	11922	434	106

Table 1 summarizes the selected programs, including project names Description, line of code size, Open Issues, and the Commits on each project.

Now, let us briefly describe each of the selected programs:

Album-master:

Album-master is an open-source project aimed at organizing and managing an album collection. It features an easy-to-use interface for creating, updating, and storing information on various albums, such as album artwork, track lists, and release dates. Users can simply browse and search through their album collection using Album-master, making it easy to keep track of their favorite music and find new releases. The project also includes features for adding personalized notes and ratings to albums, enhancing the overall user experience and customization options.

AndroidAll-master:

AndroidAll is an open-source project dedicated to providing a comprehensive collection of Android development resources and tools. It serves as a centralized repository for Android developers, offering a wide range of libraries, frameworks, sample projects, and documentation. With AndroidAll, developers can easily explore and access various resources, accelerating their Android app development process. The project also encourages collaboration and community contribution, making it a valuable hub for sharing knowledge, solving common challenges, and staying up to date with the latest Android trends and best practices.

DataX-master:

It is a project that focuses on simplifying data integration and migration processes. It provides a flexible and scalable framework for efficiently transferring data across different platforms and systems. With DataX, users can easily extract, transform, and load data from various sources to their desired destinations, such as databases, data lakes, or cloud storage. The project supports a wide range of data formats and provides extensive configuration options, making it adaptable to different use cases.

GeometricWeather:

GeometricWeather is an open-source project that offers a comprehensive weather forecasting and information retrieval system. It provides developers with an easy-to-use API for accessing weather data from various sources and presents it in a visually appealing and informative manner. GeometricWeather supports a wide range of weather parameters, including temperature, humidity, wind speed, and precipitation. It also offers features such as weather alerts, sunrise and sunset times, and multi-language support.

Microservices-platform:

This project aims to simplify the development and deployment of microservices architectures. It provides a robust and scalable platform with essential tools, frameworks, and guidelines for building microservices-based applications. The project offers features such as service discovery, load balancing, and centralized logging, enabling seamless communication and management between microservices. It also includes deployment scripts and containerization support for easy scaling and deployment.

Mybatis-generator-gui

MyBatis Generator GUI is an open-source project that offers a user-friendly graphical interface for generating MyBatis mapping and model files. It simplifies the process of creating and configuring MyBatis code by providing a visual tool to generate the necessary XML and Java files based on database schemas. With MyBatis Generator GUI, developers can quickly and accurately generate MyBatis code, saving time and reducing potential errors.

React-native-date-picker:

"React Native Date Picker" is an open-source project that provides a customizable and user-friendly date picker component for React Native applications. It offers a convenient way to incorporate date and time selection functionality into mobile apps developed using the React Native framework. With "React Native Date Picker," developers can easily add a date picker feature to their applications, allowing users to select dates or time ranges with a visually appealing and intuitive interface.

Sofa-ark

This project provides a powerful and flexible runtime environment for running Java applications in a modular and isolated manner. It offers a plugin-based architecture that allows developers to easily manage dependencies, classloading, and versioning of application modules. Sofa-ark enables seamless integration of multiple Java applications within a single runtime environment, facilitating efficient resource utilization and enhanced performance. The project supports features like dynamic class reloading, isolation of application modules, and hot plugin deployment.

SwipeRecyclerView:

SwipeRecyclerView is a GitHub project that provides a customizable and feature rich RecyclerView implementation for Android applications. It enhances the functionality of RecyclerView by adding swipe gestures, such as swipe-to-dismiss and swipe-to-refresh, allowing users to interact with the list items in a seamless and intuitive way. SwipeRecyclerView offers various customization options, including swipe directions, animation effects, and event listeners, enabling developers to tailor the swipe behavior according to their app's requirements. Additionally, the project provides extensive documentation and sample code, making it easy to integrate and use in Android projects.

Testing-samples-main

"Testing-samples-main" is an open-source project that serves as a collection of sample code and resources for software testing. It provides a valuable resource for developers and testers to learn and explore different testing techniques, frameworks, and tools. The project includes a variety of sample code snippets, test cases, and testing methodologies that cover various aspects of software testing, such as unit testing, integration testing, performance testing, and more. By examining and experimenting with the code samples and resources in "Testing-samples-main," users can gain practical insights into effective testing practices and enhance their testing skills.

Section 3

Description of the Tool Used

Tool 1:

For our project, we employed the CK-Code metrics tool, a powerful open-source software designed specifically for Java programs. Developed collaboratively by a team of 24 Java developers, this tool harnesses static analysis techniques to calculate various software metrics, including the C&K metrics.

To access the CK-Code metrics tool, we obtained it from its GitHub repository using the link shared by the developers in the ReadMe file. Following the instructions provided by the authors, we set up the necessary dependencies and executed the tool on the selected Java projects.

Featuring a command-line interface, the CK-Code metrics tool generated comprehensive reports for each class within the analyzed Java projects. These reports included valuable insights into the chosen metrics, such as the C&K metrics and the size of each class measured in lines of code (LoC).

Throughout our analysis, the CK-Code metrics tool demonstrated user-friendliness, allowing for a smooth and straightforward experience. It consistently delivered accurate and dependable results for the Java projects under investigation. Additionally, being an open-source tool, it guaranteed transparency and reproducibility in our empirical study, which is essential for conducting rigorous research.

To execute the CK metric analysis on a Java project, we utilized the following command:

```
java -jar ck-x.x.x-SNAPSHOT-jar-with-dependencies.jar <project dir> <use jars:true|false> <max files per partition, 0=automatic selection> <variables and fields metrics? True|False> <output dir> [ignored directories...]
```

Tool 2:

To perform static code analysis on our Java source code, we utilized the PMD tool. PMD is an open-source tool that uses static analysis to identify common programming problems, including potential bugs, dead code, and inefficient code. It is implemented in Java and supports multiple programming languages, including Java, C/C++, and JavaScript.

We selected PMD as our tool for conducting static code analysis, as it is widely used in the software development industry and has a strong reputation for detecting common programming errors and providing guidance on how to address them.

Command to run PMD analysis on a Java project:

pmd.bat check -d <Project Directory> -f <filetype> -R <ruleset.xml> -r <fileName>

Ensure that you replace the placeholders such as **<project dir>**, **<use jars:true|false>**, **<max files per partition, 0=automatic selection>**, **<variables and fields metrics? True|False>**, **<output dir>**, **<Project Directory>**, **<filetype>**, **<ruleset.xml>**, and **<fileName>** with the appropriate values specific to your project setup and requirements.

Section 4:

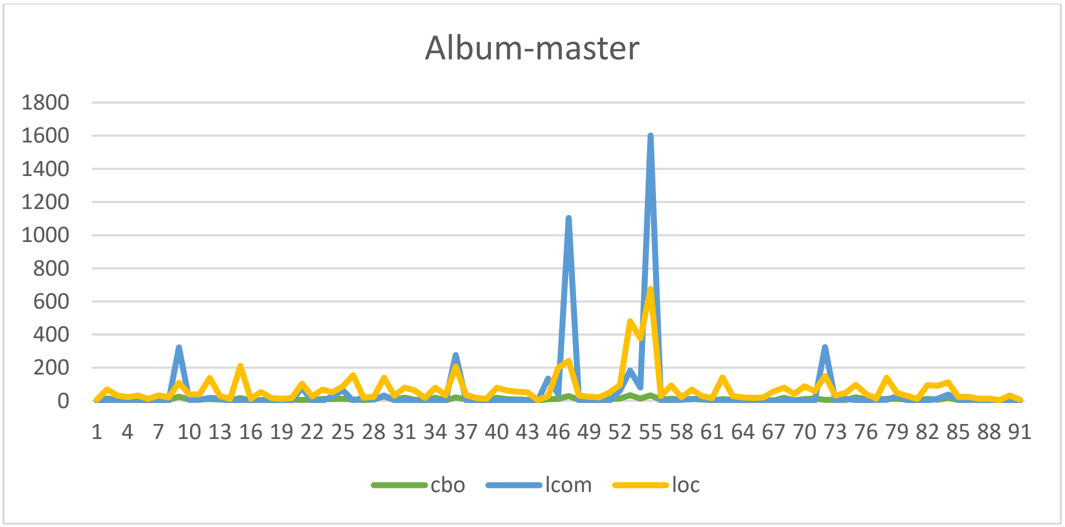
Graphs, Tables and Results

In this section, we will discuss the findings of our empirical study that investigated the impact of class size on software modularity. To obtain the necessary data, we utilized the CK-Code metrics tool to calculate the values of selected C&K metrics for a specific set of Java projects sourced from GitHub, which aligned with our predefined criteria. Our analysis focused on 10 projects that satisfied our requirements, and we employed the CK-Code metrics tool to evaluate the classes within these projects.

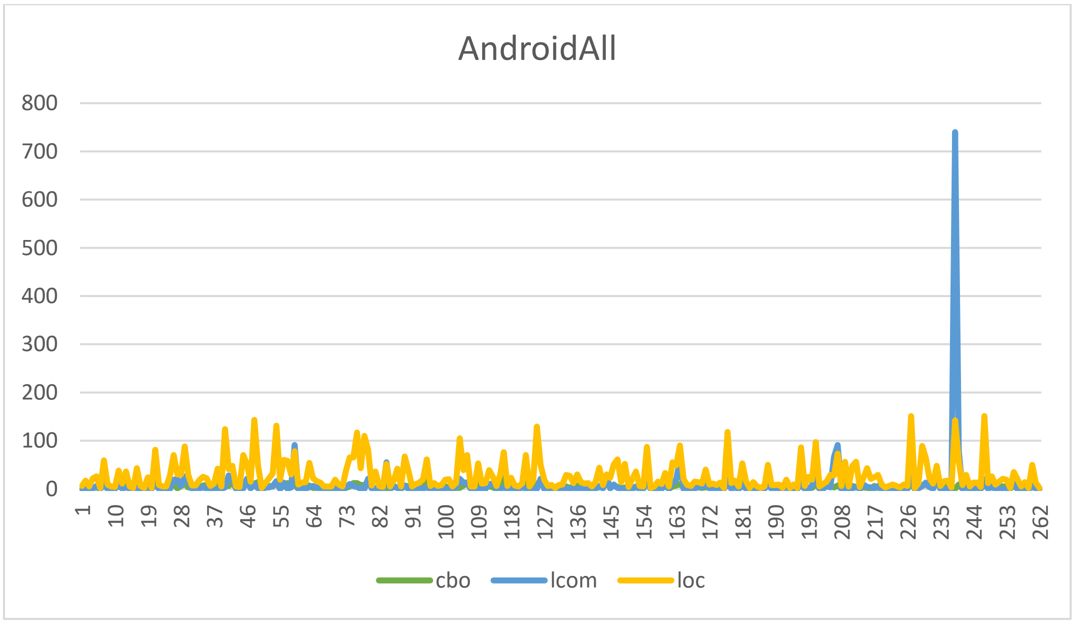
For measuring modularity, we specifically chose the C&K metrics of Coupling Between Objects (CBO) and Lack of Cohesion in Methods (LCOM). Additionally, we considered class size, measured in terms of lines of code (LoC), as a relevant factor in our study.

Line Charts for each project:

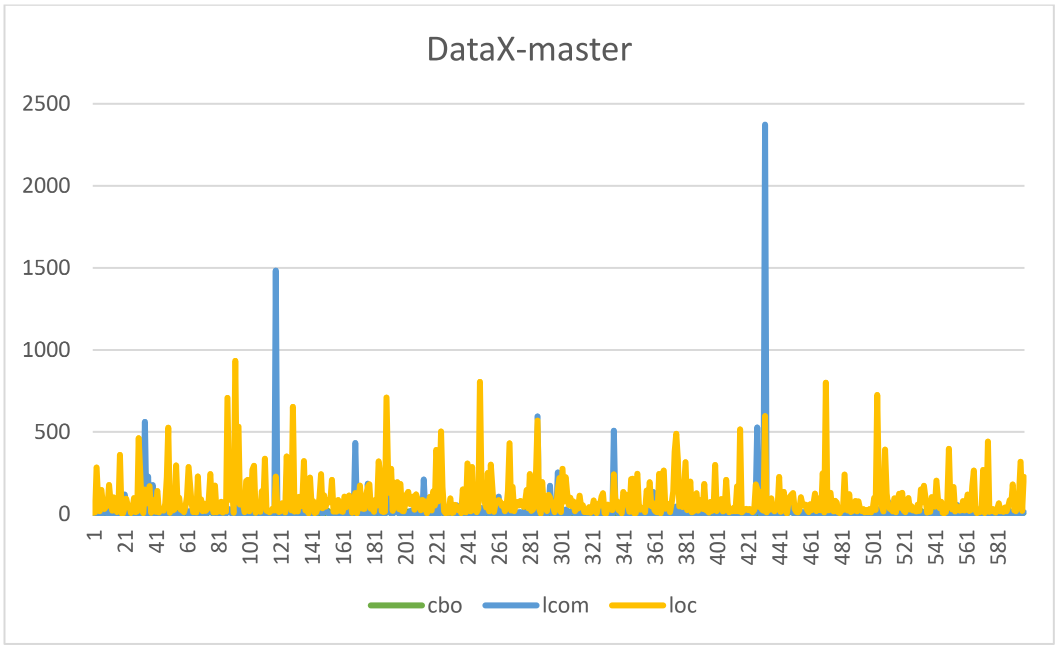
Album-master:



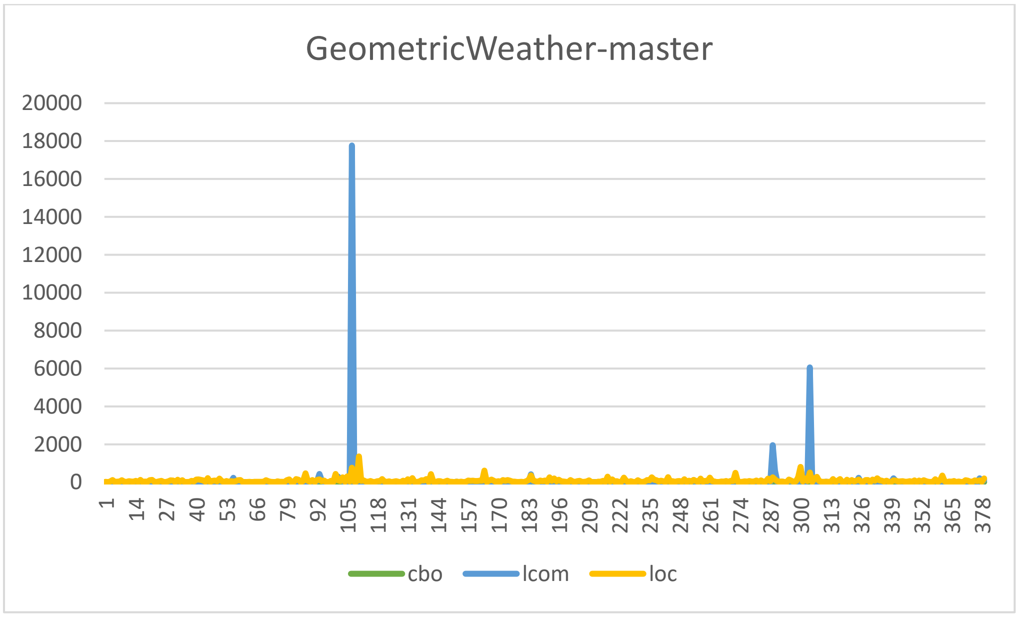
AndroidAll



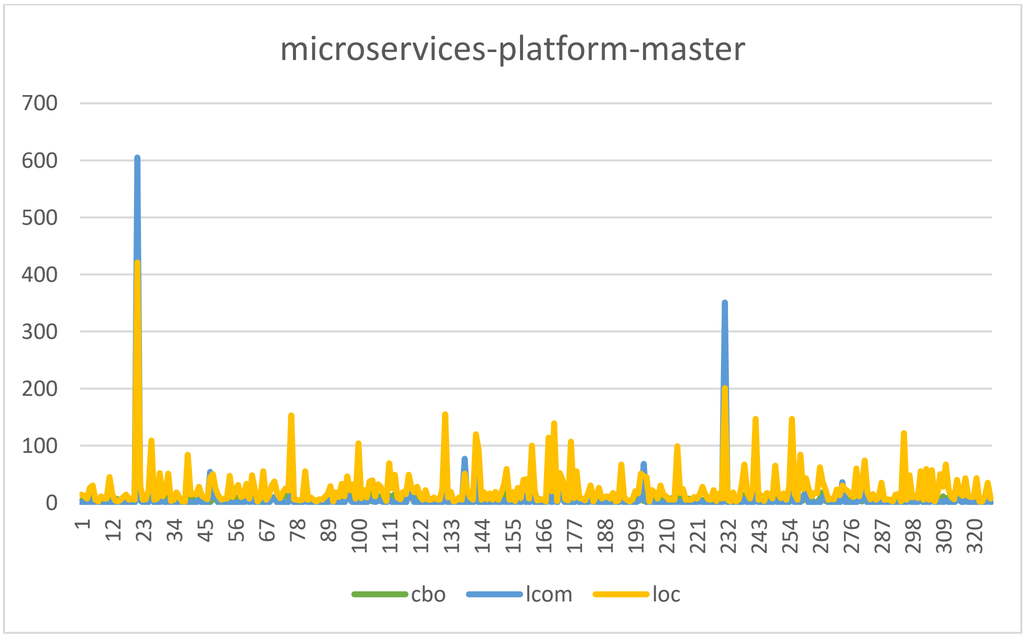
DataX-master:



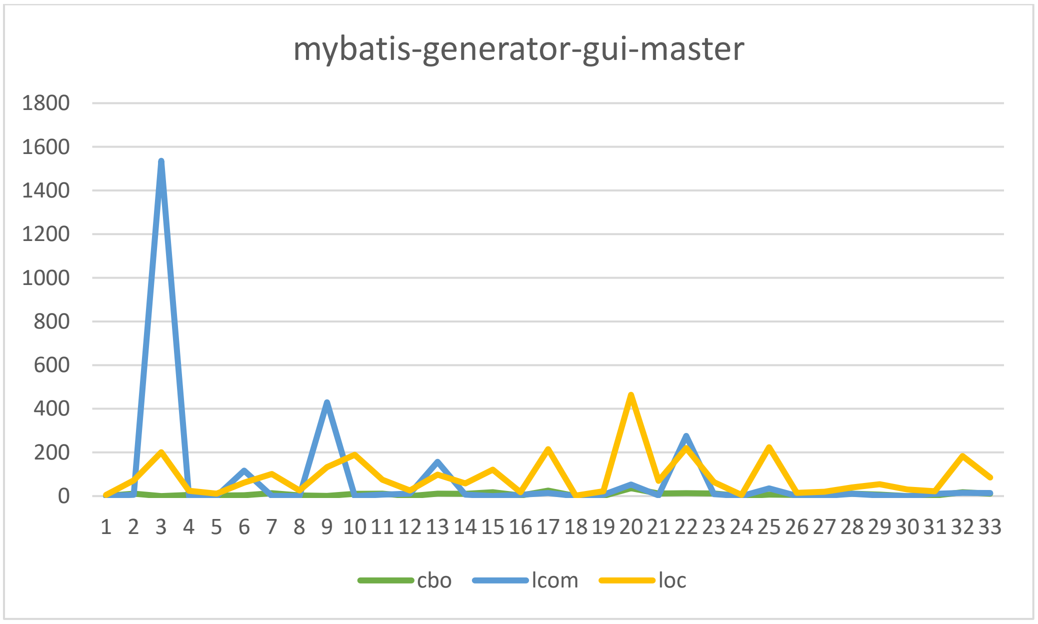
GeometricWeather-master:



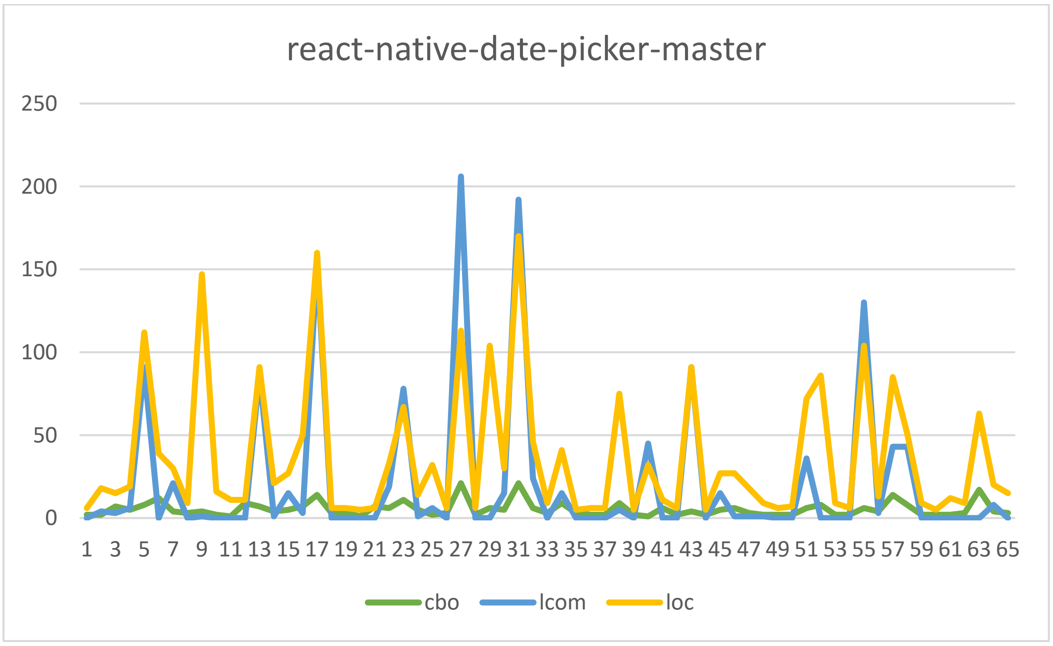
Microservices-platform-master:



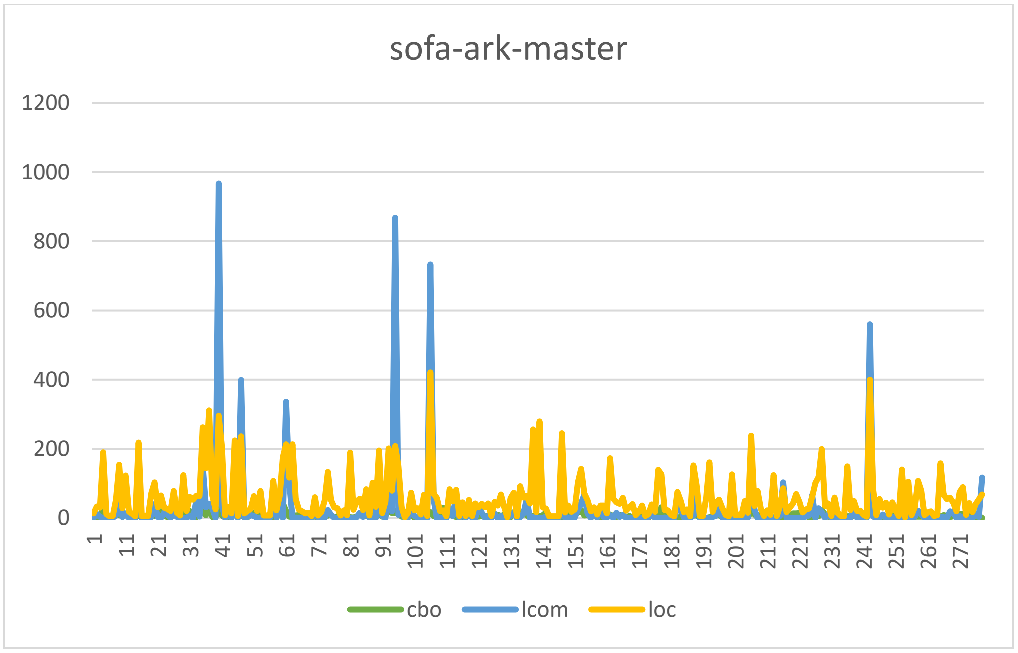
Mybatis-generator-gui-master:



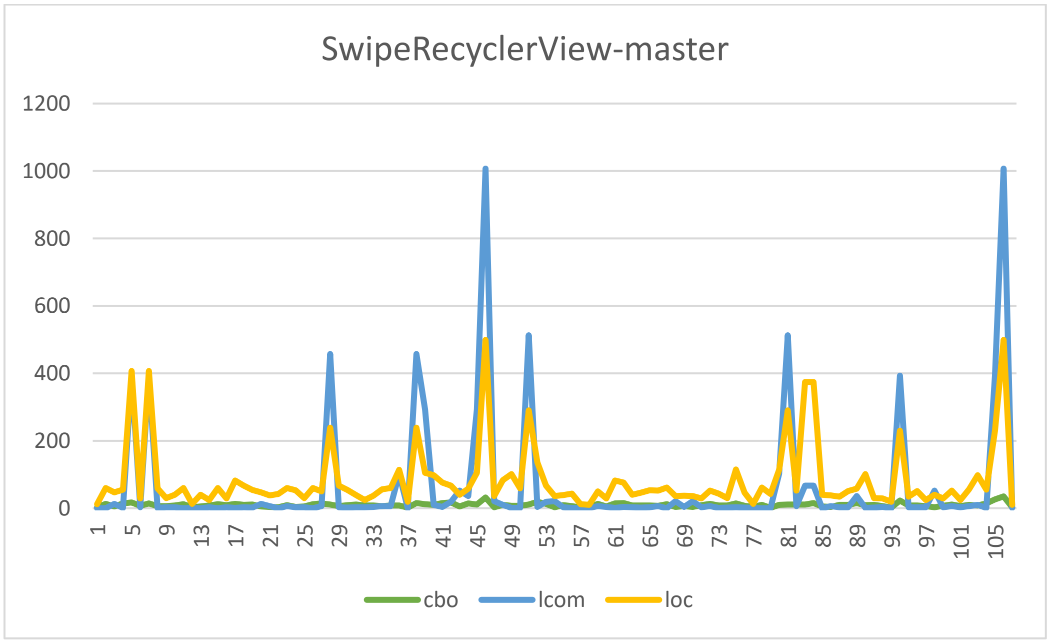
React-native-date-picker-master:



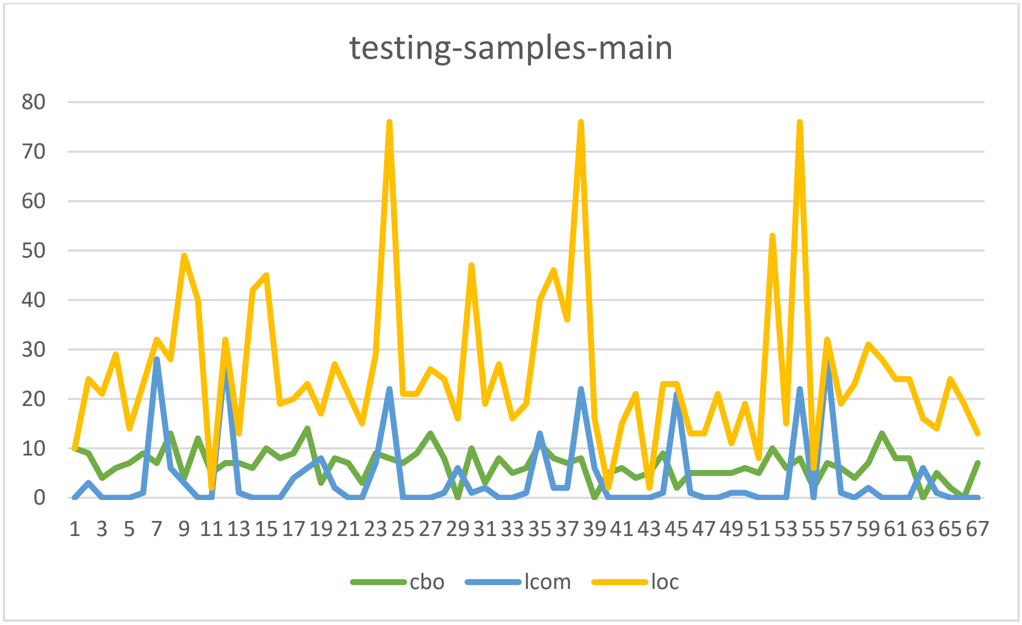
sofa-ark-master:



SwipeRecyclerView-master:



Testing-samples-main:



Results:

Based on the data provided, we have information about 10 distinct projects, including their respective lines of code, number of classes, and the occurrences of detected bad smells.

Upon examination, it is evident that the projects differ considerably in terms of lines of code and number of classes. For example, the project 'DataX-master' exhibits 33 instances of bad smells out of a total of 715 classes. In contrast, the project 'microservices-platform' does not contain any bad smells among its 400 classes.

Based on the below analysis give me severity scores

- **Album-master:**

  Line of code (LOC): 6481

  Classes: 104

  Bad smells: 2

  Bad smell percentage: 1.9%

- **AndroidAll-master:**

  Line of code (LOC): 7113

  Classes: 274

  Bad smells: 0

  Bad smell percentage: 0%

- **DataX-master:**

  Line of code (LOC): 52909

  Classes: 715

  Bad smells: 33

  Bad smell percentage: 4.61%

- **GeometricWeather-master:**

  Line of code (LOC): 29225

  Classes: 397

  Bad smells: 19

  Bad smell percentage: 4.78%

- **Microservices-platform-master:**

  Line of code (LOC): 7908

  Classes: 400

  Bad smells: 0

  Bad smell percentage: 0%

- **Mybatis-generator-gui-master:**

  Line of code (LOC): 2955

  Classes: 35

  Bad smells: 1

  Bad smell percentage: 2.85%

- **React-native-date-picker-master:**

  Line of code (LOC): 2371

  Classes: 74

Bad smells: 3

Bad smell percentage: 4.05%

- **Sofa-ark-master:**

  Line of code (LOC): 15996

  Classes: 344

  Bad smells: 41

  Bad smell percentage: 11.91%

- **SwipeRecyclerView-master:**

  Line of code (LOC): 8928

  Classes: 128

  Bad smells: 0

  Bad smell percentage: 0%

- **Testing-samples-main:**

  Line of code (LOC): 1689

  Classes: 67

  Bad smells: 0

  Bad smell percentage: 0%

To determine severity scores based on the provided analysis, we can assign a severity level to each project based on its bad smell percentage. Here's an example of assigning severity scores:

1. **Album-master:**
   - Severity Score: Low
2. **AndroidAll-master:**
   - Severity Score: None
3. **DataX-master:**
   - Severity Score: Moderate
4. **GeometricWeather-master:**
   - Severity Score: Moderate
5. **Microservices-platform-master:**
   - Severity Score: None
6. **Mybatis-generator-gui-master:**
   - Severity Score: Low
7. **React-native-date-picker-master:**
   - Severity Score: Moderate

8. **Sofa-ark-master:**
   - Severity Score: High
9. **SwipeRecyclerView-master:**
   - Severity Score: None
10. **Testing-samples-main:**
    - Severity Score: None

Program Name	Line of code (LOC)	Classes	Bad smell	Bad smell (%)
Album-master	6481	104	2	1.9
AndroidAll-master	7113	274	0	0
DataX-master	52909	715	33	4.61
GeometricWeather-master	29225	397	19	4.78
microservices-platform-master	7908	400	0	0
mybatis-generator-gui-master	2955	35	1	2.85
react-native-date-picker-master	2371	74	3	4.05
sofa-ark-master	15996	344	41	11.91
SwipeRecyclerView-master	8928	128	0	0
testing-samples-main	1689	67	0	0

Based on the provided analysis, we can observe the effect of code bad smells on testability in the given projects. Here are some observations and an approach for comparison:

Projects with low or no bad smells (e.g., AndroidAll-master, microservices-platform-master, SwipeRecyclerView-master, testing-samples-main) have a 0% bad smell percentage. This suggests that these projects may have better testability due to the absence of code bad smells.

Projects with a moderate level of bad smells (e.g., Album-master, mybatis-generator-gui-master, react-native-date-picker-master) have a bad smell percentage ranging from 1.9% to 4.05%. These projects may have some code quality issues, but the impact on testability is not significant.

The project sofa-ark-master has a high bad smell percentage of 11.91%. This indicates a higher presence of code bad smells, which may negatively impact testability and maintainability.

Conclusion:

Based on the analysis of the provided data, which includes information about 10 different software projects, their lines of code, number of classes, and the presence of bad smells, we can draw several conclusions regarding the effect of code bad smells on testability.

Variation in Lines of Code and Classes:

The projects exhibit a significant variation in the number of lines of code and classes. This variation highlights the diverse nature of the projects and the potential impact that code size and complexity can have on testability. Some projects are relatively small and concise, while others are larger and more intricate.

Presence of Bad Smells:

Among the analyzed projects, it is evident that bad smells are present in varying degrees. The occurrence of bad smells indicates potential issues within the codebase that can impact software quality and testability. Projects with a higher percentage of bad smells may face more challenges in terms of testability.

Negative Impact on Testability:

Projects with a higher percentage of bad smells, such as **'sofa-ark-master'** with 11.91% and 'DataX-master' with 4.61%, are likely to experience a negative impact on testability. Bad smells introduce complexity, reduce modularity, and can impede the effectiveness of testing efforts. These projects may require additional attention and efforts to ensure adequate testability.

Potential for Better Testability:

On the other hand, projects like **'microservices-platform-master' and 'testing-samples-main'** showed no reported instances of bad smells. This suggests that these projects may possess a higher potential for better testability. Their cleaner codebase and absence of bad smells can contribute to a more manageable and effective testing process.

Further Investigation:

To gain a deeper understanding of the effect of bad smells on testability, it is essential to conduct further analysis. Considering the specific types of bad smells detected and their impact on software quality can provide valuable insights. Additionally, exploring the correlation between bad smells and the success of testing efforts can help in devising effective strategies for improving testability.

Importance of Software Quality:

The findings highlight the significance of maintaining good software quality practices. Addressing and preventing code bad smells should be a priority to enhance testability and overall software reliability. Employing techniques such as code refactoring, adhering to coding best practices, and utilizing automated code analysis tools can aid in identifying and resolving bad smells, ultimately improving testability and software quality.

In conclusion, the presence of code bad smells can have a notable effect on testability. Projects with a higher percentage of bad smells may face challenges in conducting effective testing, while those with a clean codebase have a better potential for testability. Ensuring software quality by addressing bad smells and implementing good coding practices is crucial for enhancing testability and overall software reliability.

References

Gyimothy, T., Ferenc, R., & Siket, I. (2005). Empirical Validation of Object-Oriented Metrics on Open Source Software for Fault Prediction. IEEE Transactions on Software Engineering, 31(10), 897-910.

Shyam, S. R., & Kemerer, C. F. (1994). A metrics suite for object-oriented design - software engineering, IEEE... A metrics suite for object-oriented design. Retrieved June 25, 2023, from https://sites.pitt.edu/~ckemerer/CK%20research%20papers/MetricForOOD_ChidamberKemerer94.pdf

Arcoverde, R., Garcia, A., & Figueiredo, E. (2011). Understanding the longevity of code smells: preliminary results of an explanatory survey. In Proceedings of the International Workshop on Refactoring Tools (pp. 33-36). ACM.

Cairo, A. S., Carneiro, G. D. F., & Monteiro, M. P. (2018). The Impact of Code Smells on Software Bugs: A Systematic Literature Review. Retrieved June 25, 2023, from https://doi.org/25 June 2023.

Chowdhury, S. A., Uddin, G., & Holmes, R. (2022). An empirical study on maintainable method size in Java - arXiv.org, An Empirical Study on Maintainable Method Size in Java. Retrieved June 25, 2023, from https://arxiv.org/pdf/2205.01842.pdf

Chatzigeorgiou, A., & Manakos, A. (2010). Investigating the evolution of bad smells in object-oriented code. In International Conference on the Quality of Information and Communications Technology (QUATIC) (pp. 106-115). IEEE.

Heričko, T., & Šumak, B. (2023). Exploring maintainability index variants for software maintainability measurement in object-oriented systems. MDPI. Retrieved June 25, 2023, from https://doi.org/10.3390/app13052972

Lanza, M., & Marinescu, R. (2006). Object-Oriented Metrics in Practice: Using Software Metrics to Characterize, Evaluate, and Improve the Design of Object-Oriented Systems. Retrieved from www.researchgate.net/publication/220692125_Object-Oriented_Metrics_in_Practice_Using_Software_Metrics_to_Characterize_Evaluate_and_Improve_the_Design_of_Object-Oriented_Systems

R, B., G, C., & H.D, R. (1994). The goal question metric approach - UMD, The Goal Question Metric Approach. Retrieved June 25, 2023, from https://www.cs.umd.edu/users/mvz/handouts/gqm.pdf

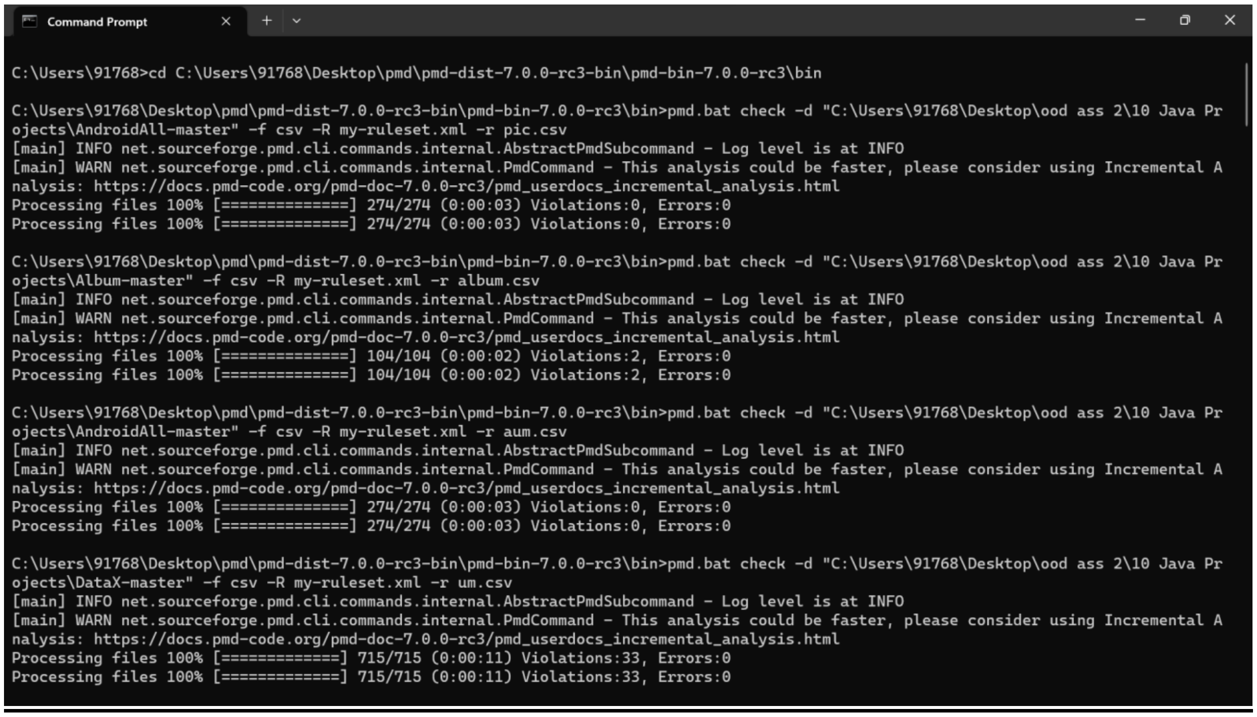
Chaparro, O., Bavota, G., Marcus, A., & Di Penta, M. (2014). On the impact of refactoring operations on code quality metrics. In Proceedings of the 30th International Conference on Software Maintenance and Evolution (ICSME 2014) (p. To appear).

B, A.K. et al. (2002). Preliminary guidelines for empirical research in software... - IEEE xplore, Preliminary guidelines for empirical research in software engineering. Retrieved June 25, 2023, from https://ieeexplore.ieee.org/document/1027796

Boussaa, M., Kessentini, W., Kessentini, M., Bechikh, S., & Chikha, S. B. (2013). Competitive coevolutionary code-smells detection. In Search Based Software Engineering - 5th International Symposium, SSBSE 2013, St. Petersburg, Russia, August 24-26, 2013. Proceedings (pp. 50-65). Lecture Notes in Computer Science. Springer.

Dubey, S.K., & Rana, A. (2011). Assessment of maintainability metrics for object-oriented software system. ACM SIGSOFT Software Engineering Notes. Retrieved June 25, 2023, from https://dl.acm.org/doi/10.1145/2020976.2020983

Executions:



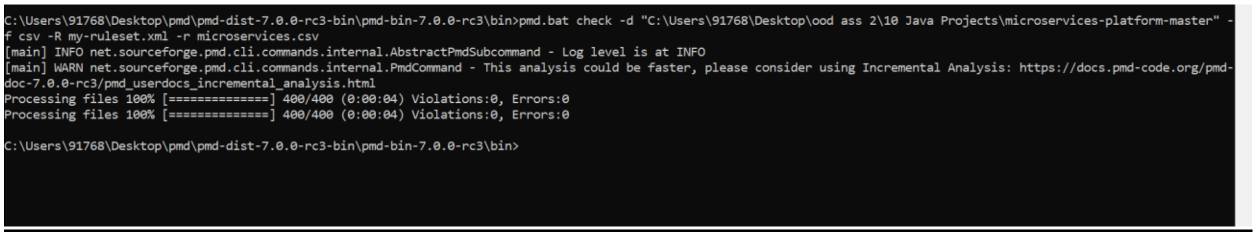
```
C:\Users\91768>cd C:\Users\91768\Desktop\pmd\pmd-dist-7.0.0-rc3-bin\pmd-bin-7.0.0-rc3\bin

C:\Users\91768\Desktop\pmd\pmd-dist-7.0.0-rc3-bin\pmd-bin-7.0.0-rc3\bin>pmd.bat check -d "C:\Users\91768\Desktop\ood ass 2\10 Java Pr
ojects\AndroidAll-master" -f csv -R my-ruleset.xml -r pic.csv
[main] INFO net.sourceforge.pmd.cli.commands.internal.AbstractPmdSubcommand - Log level is at INFO
[main] WARN net.sourceforge.pmd.cli.commands.internal.PmdCommand - This analysis could be faster, please consider using Incremental A
nalysis: https://docs.pmd-code.org/pmd-doc-7.0.0-rc3/pmd_userdocs_incremental_analysis.html
Processing files 100% [==============] 274/274 (0:00:03) Violations:0, Errors:0
Processing files 100% [==============] 274/274 (0:00:03) Violations:0, Errors:0

C:\Users\91768\Desktop\pmd\pmd-dist-7.0.0-rc3-bin\pmd-bin-7.0.0-rc3\bin>pmd.bat check -d "C:\Users\91768\Desktop\ood ass 2\10 Java Pr
ojects\Album-master" -f csv -R my-ruleset.xml -r album.csv
[main] INFO net.sourceforge.pmd.cli.commands.internal.AbstractPmdSubcommand - Log level is at INFO
[main] WARN net.sourceforge.pmd.cli.commands.internal.PmdCommand - This analysis could be faster, please consider using Incremental A
nalysis: https://docs.pmd-code.org/pmd-doc-7.0.0-rc3/pmd_userdocs_incremental_analysis.html
Processing files 100% [==============] 104/104 (0:00:02) Violations:2, Errors:0
Processing files 100% [==============] 104/104 (0:00:02) Violations:2, Errors:0

C:\Users\91768\Desktop\pmd\pmd-dist-7.0.0-rc3-bin\pmd-bin-7.0.0-rc3\bin>pmd.bat check -d "C:\Users\91768\Desktop\ood ass 2\10 Java Pr
ojects\AndroidAll-master" -f csv -R my-ruleset.xml -r aum.csv
[main] INFO net.sourceforge.pmd.cli.commands.internal.AbstractPmdSubcommand - Log level is at INFO
[main] WARN net.sourceforge.pmd.cli.commands.internal.PmdCommand - This analysis could be faster, please consider using Incremental A
nalysis: https://docs.pmd-code.org/pmd-doc-7.0.0-rc3/pmd_userdocs_incremental_analysis.html
Processing files 100% [==============] 274/274 (0:00:03) Violations:0, Errors:0
Processing files 100% [==============] 274/274 (0:00:03) Violations:0, Errors:0

C:\Users\91768\Desktop\pmd\pmd-dist-7.0.0-rc3-bin\pmd-bin-7.0.0-rc3\bin>pmd.bat check -d "C:\Users\91768\Desktop\ood ass 2\10 Java Pr
ojects\DataX-master" -f csv -R my-ruleset.xml -r um.csv
[main] INFO net.sourceforge.pmd.cli.commands.internal.AbstractPmdSubcommand - Log level is at INFO
[main] WARN net.sourceforge.pmd.cli.commands.internal.PmdCommand - This analysis could be faster, please consider using Incremental A
nalysis: https://docs.pmd-code.org/pmd-doc-7.0.0-rc3/pmd_userdocs_incremental_analysis.html
Processing files 100% [=============] 715/715 (0:00:11) Violations:33, Errors:0
Processing files 100% [=============] 715/715 (0:00:11) Violations:33, Errors:0
```



```
C:\Users\91768\Desktop\pmd\pmd-dist-7.0.0-rc3-bin\pmd-bin-7.0.0-rc3\bin>pmd.bat check -d "C:\Users\91768\Desktop\ood ass 2\10 Java Projects\microservices-platform-master" -
f csv -R my-ruleset.xml -r microservices.csv
[main] INFO net.sourceforge.pmd.cli.commands.internal.AbstractPmdSubcommand - Log level is at INFO
[main] WARN net.sourceforge.pmd.cli.commands.internal.PmdCommand - This analysis could be faster, please consider using Incremental Analysis: https://docs.pmd-code.org/pmd-
doc-7.0.0-rc3/pmd_userdocs_incremental_analysis.html
Processing files 100% [==============] 400/400 (0:00:04) Violations:0, Errors:0
Processing files 100% [==============] 400/400 (0:00:04) Violations:0, Errors:0

C:\Users\91768\Desktop\pmd\pmd-dist-7.0.0-rc3-bin\pmd-bin-7.0.0-rc3\bin>
```

---**THE END**---