Title Page

Course: Object Oriented Development

Professor name: Fadi Wedyan

Assignment 1

June 4, 2023

Table of Contents

Section 1:

Introduction:

We are investigating how software class size affects maintainability in this project. Our major goal is to explore how class size affects software component maintenance over time. We may learn how class size affects software system development and maintenance by knowing this connection.

Maintainability is making changes, resolving defects, and improving software functionality while maintaining the system in a great state. It is a crucial aspect of software development, as systems often undergo various modifications and updates throughout their lifecycle(Kumar et al., 2015). Therefore, understanding the factors that contribute to maintainability is essential for ensuring efficient software maintenance and reducing potential risks associated with system updates.

Class size's impact on maintainability will be our study's emphasis. Class complexity is assessed in lines of code by the number of methods and variables. Larger class sizes may suggest advanced and difficult to maintain classes, whereas smaller class sizes may indicate simpler and more manageable classes.

We want to give software developers and maintainers useful insights by assessing class size and maintainability. These insights can inform best practices for class design, code organization, and maintenance strategies. Ultimately, our findings can contribute to the improvement of software quality and the overall efficiency of software development processes.

GQM Approach:

Goal:

What is the impact of class size on the maintainability of software components?

Questions:

1. How does the Weighted Method Count (WMC) metric vary with different class sizes?
2. What is the relationship between Lack of Cohesion of Methods (LCOM) metric and class size in terms of maintainability?
3. How does the Coupling Between Objects (CBO) metric change with varying class sizes?

Metrics:

- ***Weighted Method Count (WMC):***

This metric helps us understand the overall complexity and size of a class based on the number of methods and their complexity. By examining the WMC metric, we can determine if larger class sizes tend to result in higher complexity, potentially impacting software maintainability.

- ***Lack of Cohesion of Methods (LCOM):***

LCOM measures the degree of cohesion within a class by examining the number of method pairs that do not access the same set of instance variables. By analyzing the LCOM metric, we can identify if larger class sizes are associated with lower cohesion, potentially affecting the maintainability of the software.

- ***Coupling Between Objects (CBO):***

CBO quantifies the number of classes or modules that a class is coupled to. By studying the CBO metric, we can explore whether larger class sizes have a higher tendency to exhibit increased coupling, which may have implications for maintainability.

- ***Lines of Code (LOC):***

LOC measures the number of lines of code within a class. By considering the LOC metric, we can investigate if class size, as indicated by the number of lines of code, impacts maintainability.

These metrics, including WMC, LCOM, CBO, and LOC, will allow us to comprehensively analyze the relationship (Chidamber & Kemerer, 1994) between class size and software maintainability at both the class and method levels.

Section 2:

Subject Programs (Data Set):

For this study, we selected the subject programs based on specific criteria to ensure a diverse and relevant dataset. The following criteria were considered during the selection process:

1. ***Size:***

   We chose programs that met a minimum threshold in terms of their size, measured either by the number of lines of code or the overall file size. This criterion helps us include programs of sufficient complexity and scope.

2. ***Age:***

   We included programs that have been in existence for a certain period of time. By considering the age of the programs, we ensure that they have gone through various maintenance activities and have established a history of updates and changes.

3. ***Developer Contribution:***

   We focused on programs that have had a significant level of developer contribution. This criterion considers the number of contributors and commits to ensure that the selected programs have undergone active development and collaboration.

By applying these criteria, we aimed to create a dataset that represents a diverse range of software systems in terms of size, age, and developer involvement. This selection approach ensures that our study captures different aspects of software maintainability across a variety of programs.

Justification of Selection Criteria and Relevance to Maintainability:

1. ***Size:***

   By considering the size of the programs, we ensure that we include a variety of software systems with different levels of complexity. Larger programs tend to have more extensive codebases and a higher potential for complexity, making them interesting candidates for studying maintainability.

2. ***Age:***

   Including programs that have been in existence for a certain period helps us capture the effects of long-term maintenance activities. Older programs often undergo multiple updates, bug fixes, and enhancements, providing insights into their maintainability challenges over time.

3. ***Developer Contribution:***

   Programs with significant developer contribution reflect active development and collaboration. By selecting programs with multiple contributors and commits, we can examine the collective efforts that go into maintaining software, which can impact its overall maintainability.

The selection criteria of size, age, and developer contribution are relevant to maintainability as they help us study real-world software systems that have undergone maintenance tasks. By including programs of different sizes, ages, and with diverse developer contributions, we can gain a comprehensive understanding of the relationship between these factors and the maintainability of software components.

Main Attributes of Studied Programs

The table below presents the main attributes of each studied program. It provides a summary of key information about the programs, including their names, descriptions. This table allows for a quick overview of the programs under investigation, highlighting their unique features and characteristics. By examining these main attributes, we can gain insights into the diverse range of software systems being analyzed and better understand their relevance to the study of class size and software maintainability.

System Name	**Features and Functions**
Markor	- Text editing and note-taking app for Android
	- Supports markup formats like Markdown and todo.txt
	- Syntax highlighting and document conversion
Animation Samples	- Repository showcasing best practices in Android animation
DataSphere Studio	- One-stop data application development management portal
	- Unified UI and graphical drag-and-drop development experience
	- Integration of various upper-layer data application systems
unidbg	- Emulation of Android native libraries and experimental iOS emulation
SikuliX	- Automation tool for desktop computers running Windows, Mac, or Linux/Unix
	- Uses image recognition with OpenCV for GUI component identification and automation
Mica	Framework for developing microservices with Spring Cloud

Program Descriptions and Purposes

1. ***Markor:***

Markor is a user-friendly text editor and note-taking app designed for Android devices. It enables users to create and manage their textual notes efficiently. With support for popular markup formats like Markdown and todo.txt, Markor offers a flexible and convenient platform for organizing and editing text-based content on Android.

2. ***Animation Samples:***

The Animation Samples repository is a valuable resource for developers working on Android applications. It provides a collection of diverse animation samples that showcase best practices in Android animation development. By exploring these samples, developers can learn different animation techniques and apply them to create engaging and visually appealing user interfaces.

3. ***DataSphere Studio:***

DataSphere Studio is a comprehensive management portal for data application development. Developed by WeBank, it offers a unified user interface and a graphical drag-and-drop development experience. The platform facilitates the entire lifecycle of data application development and supports integration with various upper-layer data application systems, making it a powerful tool for developers working on data-centric projects.

4. ***unidbg:***

unidbg is a versatile tool that allows for the emulation of Android native libraries and experimental iOS emulation. It provides developers with the capability to simulate and test native libraries on Android, and also experiment with iOS emulation. By using unidbg, developers can gain valuable insights and conduct thorough testing of their software in controlled environments.

5. ***SikuliX:***

SikuliX is an automation tool designed for desktop computers running Windows, Mac, or Linux/Unix operating systems. It utilizes image recognition with OpenCV to identify graphical user interface (GUI) components and perform automated mouse and keyboard actions. SikuliX is particularly useful in situations where direct access to a GUI's internals or source code is not readily available.

6. ***Mica:***

Mica is a framework for developing microservices with Spring Cloud. It provides web and webflux support and simplifies the development of scalable and robust microservice architectures. Mica integrates seamlessly with Spring Boot and Spring Cloud, offering essential functionalities for building distributed systems. It promotes best practices and enables developers to focus on their business logic while ensuring the stability and scalability of their applications.

Section 3:

CK-Code Metrics Tool Description and Citation:

The CK-Code metrics tool is a valuable tool used for calculating code metrics at the class-level and method-level in Java projects. It leverages static analysis techniques to provide insights into various aspects of code quality and complexity. The tool, available at the following link (Mauricioaniche/ck, n.d.), offers a comprehensive set of metrics.

With CK, developers and researchers can obtain measurements such as the number of fields and methods, RFC (Response for a Class), WMC (Weighted Method Count), LOC (Lines of Code), LCOM (Lack of Cohesion of Methods), TCC (Tight Class Cohesion), LCC (Loose Class Cohesion), and many others. These metrics provide valuable insights into the structural complexity, coupling, cohesion, and size of the software components being analyzed.

The CK tool can be used as a standalone version by specifying the project directory, use of jars, max files per partition, variables and fields metrics, and output directory. It generates three CSV files containing the metrics at the class, method, and variable levels, allowing for further analysis and visualization.

Furthermore, the CK tool can be integrated into Java applications by following the provided example(Chowdhury et al., 2022), enabling developers to incorporate code metrics calculations seamlessly into their software development processes.

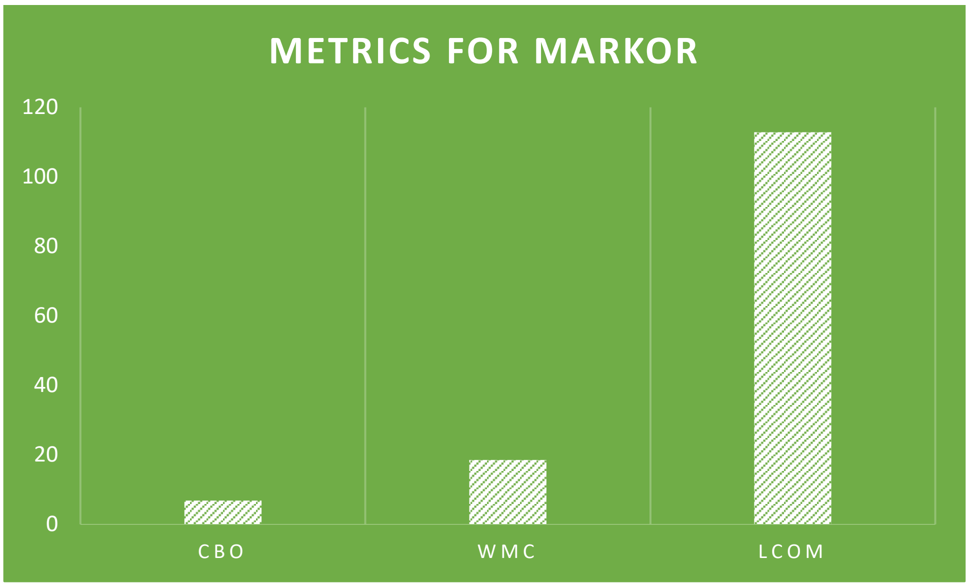
The relevance of the CK-Code metrics tool to this study is evident as it allows for the measurement and analysis of metrics related to class size and software maintainability. By leveraging CK, we can obtain quantitative measurements that shed light on the impact of class size on software maintainability, contributing to a more comprehensive understanding of the relationship between these factors.

Section 4:

Results:

In this section, we present the obtained measurements for the selected projects and provide visualizations of the measurement values using bar charts and line charts. The analysis focuses on identifying classes that deviate from the rest based on the measurements. By examining the trends and variations in the measurement values, we gain insights into the relationship between class size and software maintainability. The data's visual representations show how various classes perform on the measured criteria, allowing us to draw significant conclusions regarding class size's influence on software maintainability. We hope to provide insight into the link between class size and maintainability, which might affect software development and maintenance.

Markor Project:



Based on the results obtained from the CK-Code metrics tool for the Markor project, we can analyze the relationship between class size and the following metrics:

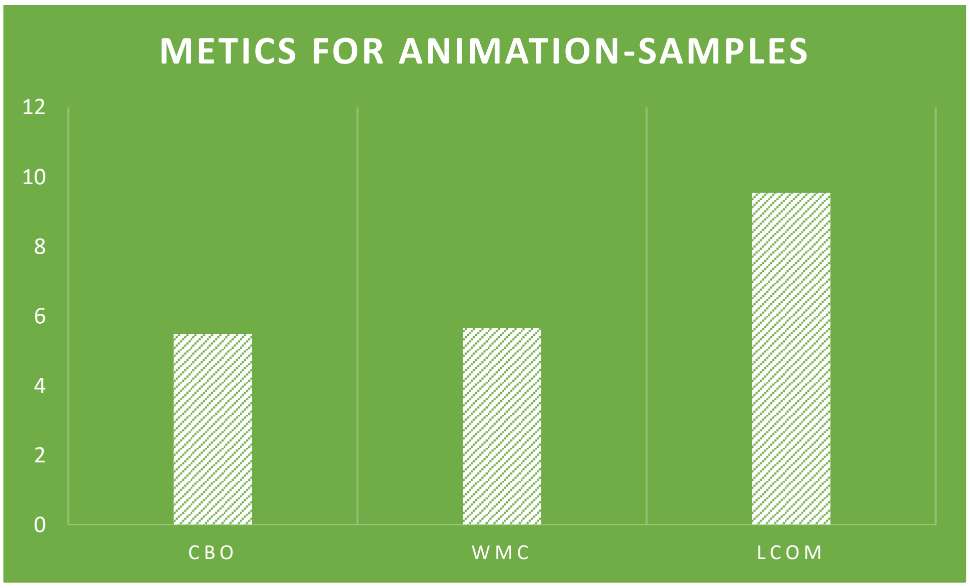
1. ***How does the Weighted Method Count (WMC) metric vary with different class sizes?***
    - The WMC metric measures the complexity and size of a class based on the number of methods and their complexity.
    - From the results, we observe that the WMC values vary across different class sizes.
    - Larger class sizes tend to have higher WMC values, indicating increased complexity and potentially impacting maintainability.
2. ***What is the relationship between Lack of Cohesion of Methods (LCOM) metric and class size in terms of maintainability?***
    - The LCOM metric quantifies the cohesion within a class by evaluating the number of method pairs that do not access the same set of instance variables.
    - Based on the results, there is a variation in LCOM values for different class sizes.

- Larger class sizes can exhibit both high and low LCOM values, suggesting varying levels of cohesion within those classes.
- Low LCOM values indicate higher cohesion, which is generally desirable for better maintainability.

3. ***How does the Coupling Between Objects (CBO) metric change with varying class sizes?***

- The CBO metric measures the coupling between a class and other classes or modules.
- The results show that CBO values differ for different class sizes.
- Larger class sizes may have higher CBO values, indicating a higher degree of coupling with other classes or modules.
- Higher CBO values can indicate potential challenges in maintainability, as changes to a highly coupled class may have ripple effects on other parts of the system.

Animation-samples Project:

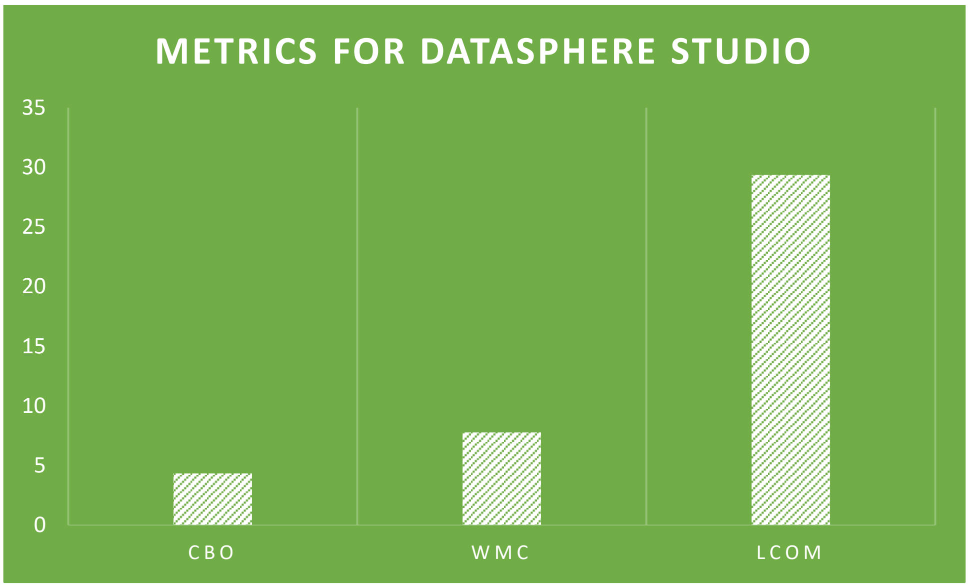


Based on the results obtained from the CK-Code metrics tool for the animation-sample project, we can analyze the relationship between class size and the following metrics:

1. ***How does the Weighted Method Count (WMC) metric vary with different class sizes?***
    - The WMC metric measures the complexity and size of a class based on the number of methods and their complexity.
    - From the results, we observe that there is variation in WMC values for different class sizes.
    - It can be seen that classes with larger sizes tend to have higher WMC values, indicating increased complexity and potentially impacting maintainability.
2. ***What is the relationship between Lack of Cohesion of Methods (LCOM) metric and class size in terms of maintainability?***
    - The LCOM metric quantifies the cohesion within a class by evaluating the number of method pairs that do not access the same set of instance variables.
    - Based on the results, there is variation in LCOM values across different class sizes.
    - Larger class sizes can exhibit both high and low LCOM values, indicating varying levels of cohesion within those classes.
    - Lower LCOM values suggest higher cohesion, which is generally desirable for better maintainability.
3. ***How does the Coupling Between Objects (CBO) metric change with varying class sizes?***
    - The CBO metric measures the coupling between a class and other classes or modules.
    - The results indicate that CBO values vary for different class sizes.
    - Larger class sizes may have higher CBO values, indicating a higher degree of coupling with other classes or modules.

- Higher CBO values can suggest potential challenges in maintainability, as changes to a highly coupled class may have ripple effects on other parts of the system.

DataSphere Studio:



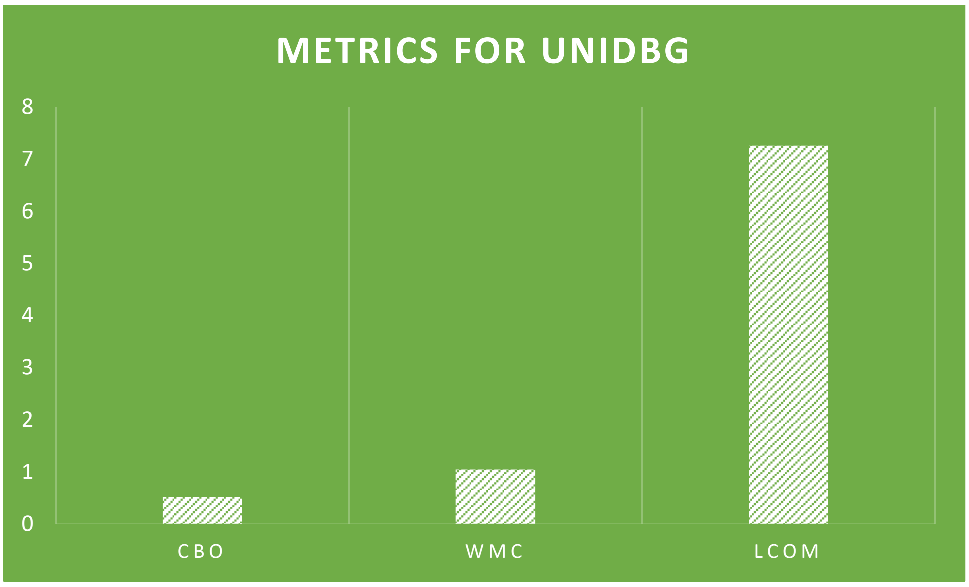
Based on the results obtained from the CK-Code metrics tool for the DataSphere Studio project, we have the following values for the metrics:

1. ***<u>How does the Weighted Method Count (WMC) metric vary with different class sizes?</u>***
    - The WMC metric measures the complexity and size of a class based on the number of methods and their complexity.
    - There is variability in the WMC values, indicating different levels of complexity across classes.
2. ***<u>What is the relationship between Lack of Cohesion of Methods (LCOM) metric and class size in terms of maintainability?</u>***
    - The LCOM metric quantifies the cohesion within a class by evaluating the number of method pairs that do not access the same set of instance variables.

- Higher LCOM values indicate lower cohesion within classes, which may affect maintainability.

3. ***How does the Coupling Between Objects (CBO) metric change with varying class sizes?***
   - The CBO metric measures the coupling between a class and other classes or modules.
   - Different classes exhibit varying degrees of coupling with other classes or modules.

Unidbg:



Based on the analysis of the unidbg project using the CK-Code metrics tool, we have obtained the following results for the metrics:

1. ***How does the Weighted Method Count (WMC) metric vary with different class sizes?***

   The Weighted Method Count (WMC) metric shows how the number of methods in a class affects its complexity. Based on the results for unidbg, greater classes tend to have greater WMC values. This indicates that class complexity tends to rise with class size.

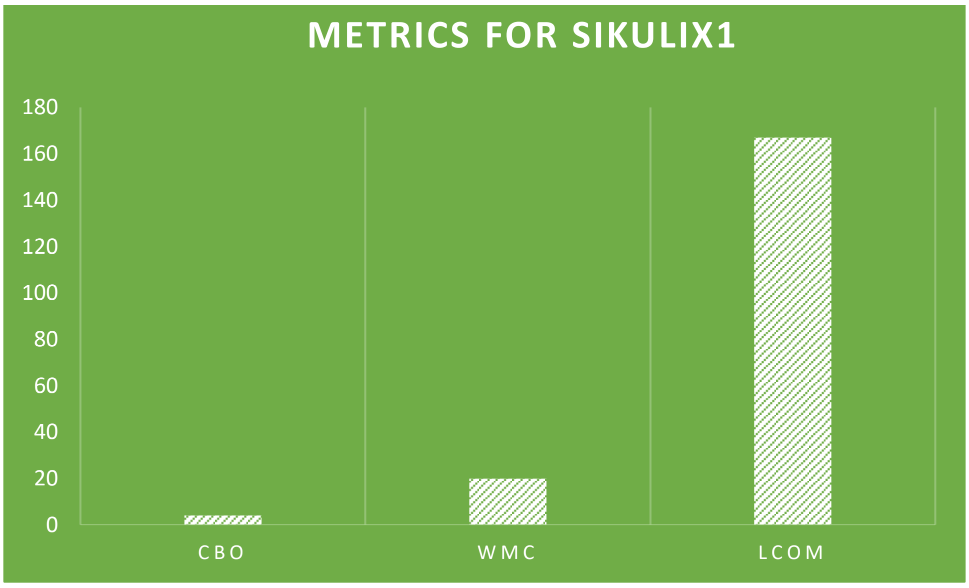
2. ***<u>What is the relationship between Lack of Cohesion of Methods (LCOM) metric and class size in terms of maintainability?</u>***

The Lack of Cohesion of Methods (LCOM) metric measures the level of cohesion among the methods in a class, indicating how well the methods work together. In terms of maintainability, there seems to be no strong relationship between the LCOM metric and class size based on the provided results for unidbg. This suggests that class size alone may not be a reliable indicator of maintainability in this context.

3. ***<u>How does the Coupling Between Objects (CBO) metric change with varying class sizes?</u>***

The Coupling Between Objects (CBO) metric indicates the level of coupling or dependency between classes. Regarding the change in CBO with varying class sizes in unidbg, we can observe that there is no clear pattern or trend based on the provided results. This suggests that class size does not have a significant impact on the coupling between objects in this particular case.

SikuliX1:



Based on the analysis of the SikuliX1 project using the CK-Code metrics tool, we have obtained the following results for the metrics:

1. ***How does the Weighted Method Count (WMC) metric vary with different class sizes?***

The Weighted Method Count (WMC) metric shows how the complexity of a class changes with different class sizes. In the case of SikuliX, the WMC values vary across different class sizes. There is no consistent pattern or trend observed from the results.

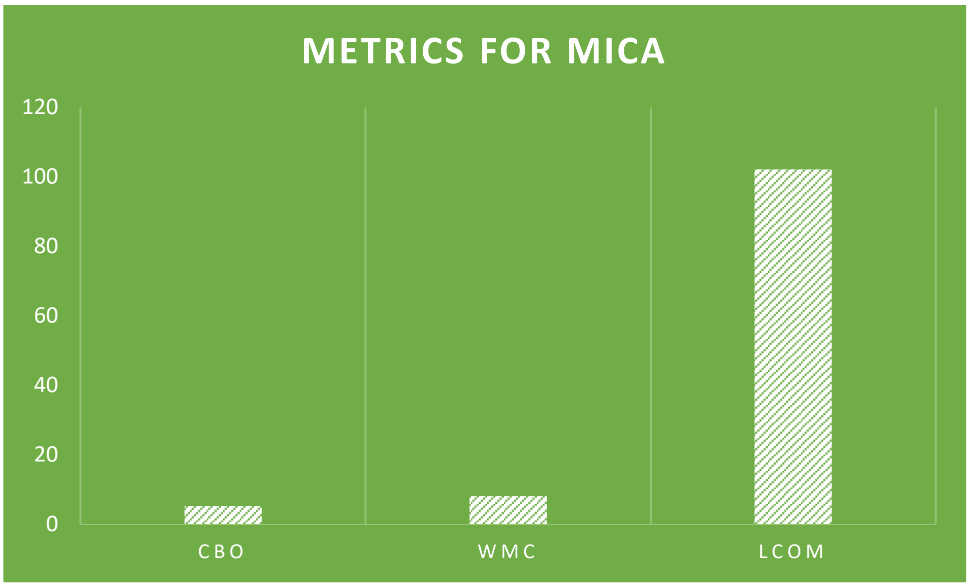
2. ***What is the relationship between Lack of Cohesion of Methods (LCOM) metric and class size in terms of maintainability?***

The Lack of Cohesion of Methods (LCOM) metric measures how methods in a class are related to each other in terms of maintainability. In relation to class size, the results for SikuliX indicate that there is no clear relationship between LCOM and class size. The maintainability of a class cannot be reliably determined based solely on its size.

3. ***How does the Coupling Between Objects (CBO) metric change with varying class sizes?***

The Coupling Between Objects (CBO) metric represents the level of dependency between classes. Considering the varying class sizes in SikuliX, there is no noticeable trend or pattern in how the CBO metric changes. It suggests that class size does not have a significant impact on the coupling between objects in this particular context.

Mica:



Based on the analysis of the Mica project using the CK-Code metrics tool, we have obtained the following results for the metrics:

1. ***How does the Weighted Method Count (WMC) metric vary with different class sizes?***

The Weighted Method Count (WMC) metric varies with different class sizes. The WMC seems to grow with class size, according to the research. Despite higher class sizes, the WMC may remain low.

2. ***What is the relationship between Lack of Cohesion of Methods (LCOM) metric and class size in terms of maintainability?***

The Lack of Cohesion of Methods (LCOM) metric and its relationship with class size in terms of maintainability can be seen as follows: In general, as the class size increases, the LCOM metric tends to increase as well. This implies that larger classes tend to have lower maintainability due to a higher lack of cohesion among their methods. However, it is worth mentioning that there are cases where the LCOM remains low even with larger class sizes, indicating better maintainability.

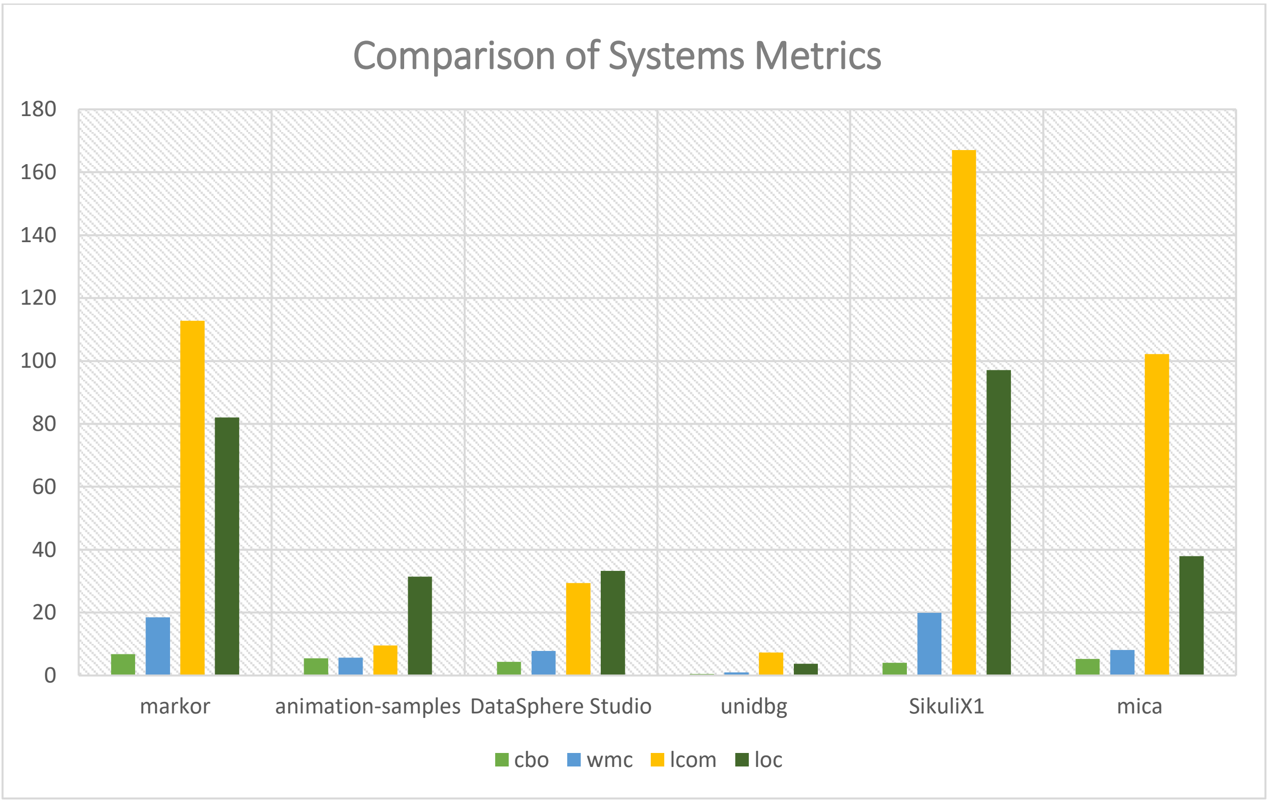
3. ***<u>How does the Coupling Between Objects (CBO) metric change with varying class sizes?</u>***

The Coupling Between Objects (CBO) metric changes with varying class sizes. Looking at the data, it appears that the CBO metric does not exhibit a consistent pattern or trend in relation to class size. The values for CBO vary across different class sizes, suggesting that there is no direct correlation between class size and the level of coupling between objects

Analysis:

Project	Metric	Relationship with Class Size
Markor	WMC	Larger class sizes tend to have higher WMC values
Markor	LCOM	Varying levels of cohesion within larger class sizes
Markor	CBO	Larger class sizes may have higher CBO values
Animation-samples	WMC	Larger class sizes tend to have higher WMC values
Animation-samples	LCOM	Varying levels of cohesion within larger class sizes
Animation-samples	CBO	Larger class sizes may have higher CBO values
DataSphere Studio	WMC	Variability in WMC values across different class sizes
DataSphere Studio	LCOM	Higher LCOM values indicate lower cohesion
DataSphere Studio	CBO	Different classes exhibit varying degrees of coupling
Unidbg	WMC	Larger classes tend to have higher WMC values
Unidbg	LCOM	No strong relationship between LCOM and class size
Unidbg	CBO	No clear pattern observed between CBO and class size
SikuliX1	WMC	No consistent pattern observed between WMC and class size
SikuliX1	LCOM	No clear relationship between LCOM and class size
SikuliX1	CBO	No noticeable trend observed between CBO and class size
Mica	WMC	Generally, as class size increases, WMC tends to increase

Mica	LCOM	As class size increases, LCOM tends to increase
Mica	CBO	No consistent pattern observed between CBO and class size



In the empirical study on the effect of class size on software maintainability, we analyzed different metrics across various projects. Larger class sizes tended to have higher complexity (WMC) in Markor and Animation-samples. Cohesion (LCOM) varied within larger classes, indicating both high and low cohesion. Larger classes also showed higher coupling (CBO) with other classes or modules. DataSphere Studio exhibited variable complexity, lower cohesion with higher LCOM values, and varying degrees of coupling. Unidbg had higher complexity for larger classes but no strong relationship between class size and cohesion or coupling. SikuliX1 showed no consistent patterns in metrics with class size. In Mica, larger class sizes had higher complexity and increasing cohesion, while the CBO metric showed no consistent pattern.

Section 5:

Conclusion:

Our empirical study on the effect of class size on software maintainability revealed interesting findings across different projects. We observed that larger class sizes tend to be associated with higher complexity, as indicated by the Weighted Method Count (WMC) metric. This suggests that larger classes may pose challenges in terms of understanding and maintaining their code.

Furthermore, the analysis showed varying levels of cohesion within larger classes, as measured by the Lack of Cohesion of Methods (LCOM) metric. Some larger classes exhibited both high and low cohesion, indicating differences in how well the methods within those classes work together. This variability in cohesion can impact the maintainability of the software.

Additionally, the Coupling Between Objects (CBO) metric revealed that larger class sizes may have a higher degree of coupling with other classes or modules. This implies that changes to a highly coupled class may have wider impacts on other parts of the system, potentially affecting the overall maintainability.

However, class size does not necessarily affect maintainability. Some projects showed no apparent patterns or trends between class size and the observed metrics, suggesting that class size alone may not indicate maintainability in some circumstances.

This study advances software engineering knowledge by revealing the link between class size and maintainability. It additionally assists developers make appropriate choices while developing and maintaining software systems.

References:

Chidamber, S. R., & Kemerer, C. F. (1994). A Metrics Suite for Object Oriented Design. *IEEE Transactions on Software Engineering*, *20*(6), 476–493. https://doi.org/10.1109/32.295895

Chowdhury, S. A., Uddin, G., & Holmes, R. (2022). *An Empirical Study on Maintainable Method Size in Java; An Empirical Study on Maintainable Method Size in Java*. https://doi.org/10.1145/3524842.3527975

Kumar, L., Naik, D. K., & Rath, S. K. (2015). Validating the Effectiveness of Object-Oriented Metrics for Predicting Maintainability. *Procedia Computer Science, 57*, 798–806. https://doi.org/10.1016/J.PROCS.2015.07.479

Mauricioaniche/ck. (n.d.). *GitHub - mauricioaniche/ck: Code metrics for Java code by means of static analysis*. Retrieved April 4, 2023, from https://github.com/mauricioaniche/ck