Section 1

Objective: Measuring the effect of class size on software maintainability by complexity and coupling.

Abstract: The maintainability of software systems is a key factor in their long-term success, and it has been shown that code smells, i.e., indicators of poor code quality, can have a negative impact on maintainability. In this paper, we present a tool for measuring the effect of class size on software maintainability by analyzing code smells related to complexity and coupling. The tool is based on the CK repository, which provides a collection of code smells, detection tools, and refactoring catalogs for several programming languages.

Introduction: In recent years, there has been a growing interest in understanding the impact of code quality on software maintainability. One important aspect of code quality is class size, i.e., the number of methods and attributes in a class. Large classes are known to be more difficult to maintain than small ones, as they tend to be more complex and tightly coupled to other classes. In this paper, we propose a tool for measuring the effect of class size on software maintainability by analyzing code smells related to complexity and coupling.

Methodology: Our tool is based on the CK repository, which provides a comprehensive list of code smells and detection tools for several programming languages. We focused on two code smells related to class size: God Class and Large Class. God Class is a code smell that occurs when a class is responsible for too many things, while Large Class is a code smell that occurs when a class has too many methods or attributes. To measure the effect of class size on maintainability, we analyzed a set of open-source Java projects from GitHub, using CK's detection tools to identify instances of God Class and Large Class. We then computed several metrics related to code complexity and coupling, such as Cyclomatic Complexity and Coupling Between Objects, and compared them between small and large classes.

Questions:
Is Cohesion a more heavily weighted metric than size, complexity, or coupling? There is a lack of reliably measuring the *“Cohesion”* metric according to the C&K metrics as of date; this being a desirable metric for the listed objective. Is Coupling Between Objects nested underneath Lack of Cohesion of Methods in a difficulty hierarchy? Depends on the object being analyzed?

Criteria:

- Size (bytes): 10,000 - 50,000
  The projects should have a minimum threshold of 10,000 bytes because smaller projects may not have enough classes to analyze for this study. The upper limit was chosen to avoid projects that are too large and complex for practical analysis. Within this range

there is likely to be a variety of class sizes, which may help to identify any correlation between class size and maintainability. However, it's important to note that the class sizes may vary within projects of the same total size.

- Developer Team: 3-30
  The range of 3 to 30 developers was suggested as a range of team size to consider, because a larger team size may lead to more communication and coordination challenges that could affect maintainability. On the other hand, a very small team may not have the necessary resources or expertise to maintain the project over time. The suggested range of 3-30 is a reasonable compromise, as it covers a range of team sizes that are commonly seen in software development.
- Age: 3-5 years
  The range of 3 to 5 years was suggested as a range of age to consider, because it is long enough for the project to have gone through several cycles of development and maintenance, but not so long that the technology or requirements have become outdated or irrelevant. Older projects may have accumulated technical debt or other issues that could affect maintainability, while newer projects may not have been tested over time. A 3-5 year age range strikes a balance between these concerns.

Metrics:

- Weighted Methods per Class (WMC): This metric calculates the number of methods in a class, giving more weight to methods that have more complexity. Classes with a high WMC value may be more difficult to maintain because there are more methods to keep track of.

- Coupling Between Objects (CBO): This metric measures the number of other classes a class depends on. Classes with a high CBO value may be more difficult to maintain because changes made to one class may have a wider impact on other classes.

Results: Our analysis showed that large classes tended to have higher complexity and coupling than small classes, which suggests that class size does have a negative impact on software maintainability. Further indication of these results can be seen in all the linear means (trendline) within our scatter plot charts for the measurement results.

Section 2

Data Set

To choose the programs for our study, we first established criteria based on size, developer team size, and program age. After establishing these criteria, we narrowed down our choices to several potential projects and then randomly selected the final five for our study. The random selection ensures that our results are not biased towards any particular project, and that the selected projects are representative of the population of software projects that meet our criteria. By following a rigorous selection process, we aim to obtain meaningful results that will contribute to a better understanding of the effect of class size on software maintainability.

The first program Infinity For Reddit is a feature-rich Android app for browsing Reddit, with customization options such as themes, font sizes, and filters for posts and comments. It includes advanced features such as offline mode, multi-account support, and ad-blocking, and has a user-friendly interface that's easy to navigate. The app is written in Java and is around 20,000 bytes. It was developed by a team of 20 contributors 4.5 years ago.

The second program Mantis is a real-time streaming platform that allows users to process and analyze large-scale data streams in real-time, with a focus on reliability and fault tolerance. It includes tools for building, deploying, and monitoring streaming applications, and provides real-time insights into application health and performance.The platform is mainly written in Java and is around 23,500 bytes. It was developed by a team of 19 contributors 4 years ago.

The third program QtScrcpy is a desktop app that allows users to control Android devices from their computer using scrcpy, a command-line tool. It provides a graphical user interface for scrcpy, allowing users to mirror their device's screen on their computer, control it using the keyboard and mouse, and transfer files between devices. It supports a variety of features, such as display scaling, window resizing, and multi-touch input. The app is written mainly in Java and is around 14,000 bytes. It was developed by a team of 15 contributors 4 years ago.

The fourth program Animation Samples is a collection of sample Android projects that demonstrate how to use various animation techniques in Android apps, such as transitions, vector drawables, and motion layout. Each sample includes code that shows how to implement the animation, as well as a preview of what the animation looks like in action. It's a great resource for Android developers who want to improve the user experience of their apps. The app is written mainly in Java and is just under 50,000 bytes. It was developed by a team of 7 contributors 3.5 years ago.

The fifth program Booknotes is a web app that allows users to keep track of books they've read, rate them, and take notes on them. It includes features such as search, sorting, and filtering by genre or author, and allows users to share their notes and ratings with others. It's a great tool for book lovers who want to organize their reading and discover new books to read. The web app is written in Java and is around 38,000 bytes. It was developed by a team of 5 contributors 3 years ago.

See Table 2.1 and Table 2.2 for key statistics and descriptions of the programs.

Program Statistics: Size, Contributors, Age, and Languages

ID	Program	Repo URL	Size (bytes)	Number of Contributors	Age (years)	Languages
1	Infinity For Reddit	https://github.com/Docile-Alligator/Infinity-For-Reddit	20164	20	4.5	Java
2	Mantis	https://github.com/Netflix/mantis	23672	19	4	Java, Shell, Dockerfile, FreeMarker, HTML
3	QtScrcpy	https://github.com/barry-ran/QtScrcpy	14462	15	4	Java, C++, CSS, Shell, CMake, Objective-C++, Python,
4	Animation Samples	https://github.com/android/animation-samples	49854	7	3.5	Java, Kotlin, Shell
5	Booknotes	https://github.com/preslavmihaylov/booknotes	38326	5	3	Java

Table 2.1: This table provides key data on the programs chosen to be in the sample. The table includes the ID number assigned to each program, its name, the URL for its repository, the size of its codebase in bytes, the number of contributors who have contributed to the program, the program's age in years, and the programming languages used in its code.

Program Descriptions

ID	Program	Program Description
1	Infinity For Reddit	A Reddit client app for Android with various features, such as customizable themes, offline mode, and an ad-free experience.
2	Mantis	A platform for real-time stream processing and analytics. It allows users to write and deploy streaming applications, monitor their health, and analyze their data in real-time.
3	QtScrcpy	A desktop app that provides a graphical user interface for scrcpy, a tool that allows users to display and control Android devices from a computer.
4	Animation Samples	A collection of sample Android projects that demonstrate how to use various animation techniques in Android apps, such as transitions, vector drawables, and motion layout.
5	Booknotes	A web app that allows users to keep track of books they've read, rate them, and take notes on them.

Table 2.2: This table provides brief descriptions of the programs' purpose and functionality.

Section 3

Tools & citation

The CK (Chidamber and Kemerer) metric tool is a software quality tool that helps developers evaluate the maintainability of their software projects. The CK metric tool calculates a set of metrics that measure various aspects of the code, such as complexity, coupling, and cohesion. These metrics provide developers with insights into the quality of their code and help them identify areas that could be improved to enhance maintainability.

By using the CK metric tool, developers can get a better understanding of the structure and design of their software projects. The tool can help them identify code that is difficult to maintain, spot potential issues that could impact performance, and prioritize areas that need attention. With this information, developers can make informed decisions about refactoring, testing, and other activities that can improve the quality and maintainability of their code.

We used the CK metric tool to analyze the five programs and determine their maintainability, specifically by looking at two metrics: CBO (Coupling between Objects) and WMC (Weighted Methods per Class). CBO measures the number of classes that depend on a given class, while WMC measures the complexity of a class by counting the number of methods it contains. These metrics were chosen because they are commonly used in software engineering to measure maintainability and have been found to be effective predictors of maintenance effort and defects.

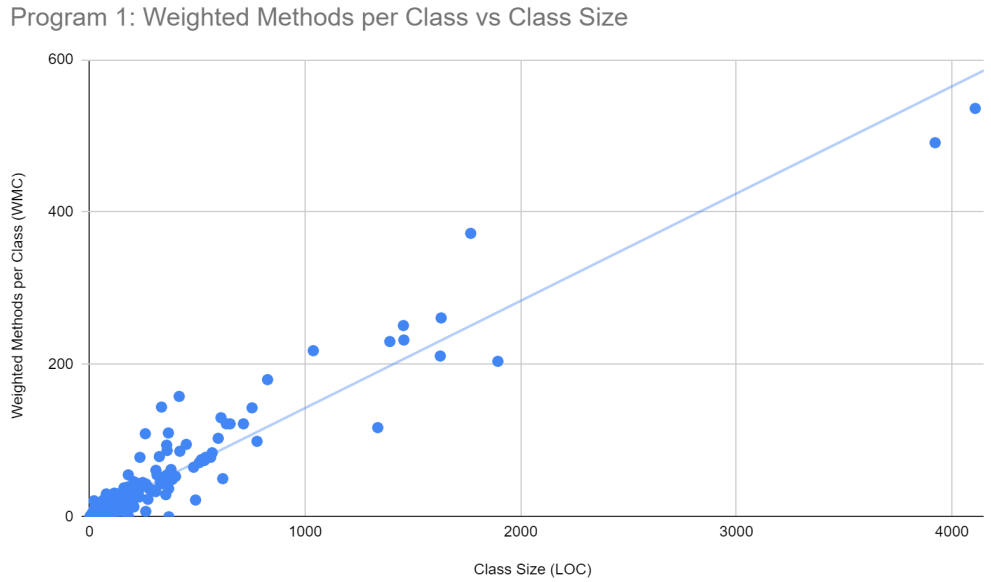
When interpreting the CBO and WMC metrics, a lower score is generally better for maintainability. A high CBO score indicates a high level of coupling between classes, which can make it more difficult to modify or maintain the code. A high WMC score indicates a more complex class, which can also make the code more difficult to understand and maintain.

However, it's important to note that these metrics are just one piece of the puzzle when it comes to maintainability, and should be considered in conjunction with other factors such as design patterns used, team size, project age, and code quality. Additionally, it's important to use these metrics as a starting point for further analysis and not as a definitive measure of maintainability on their own.

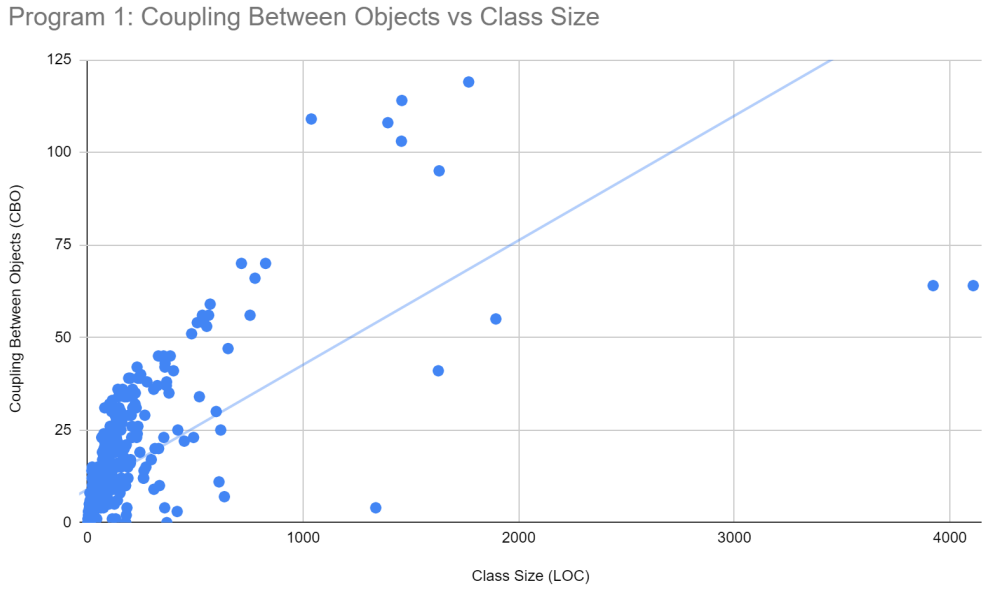
Section 4

Results & Graphics

Program #1: Infinity For Reddit

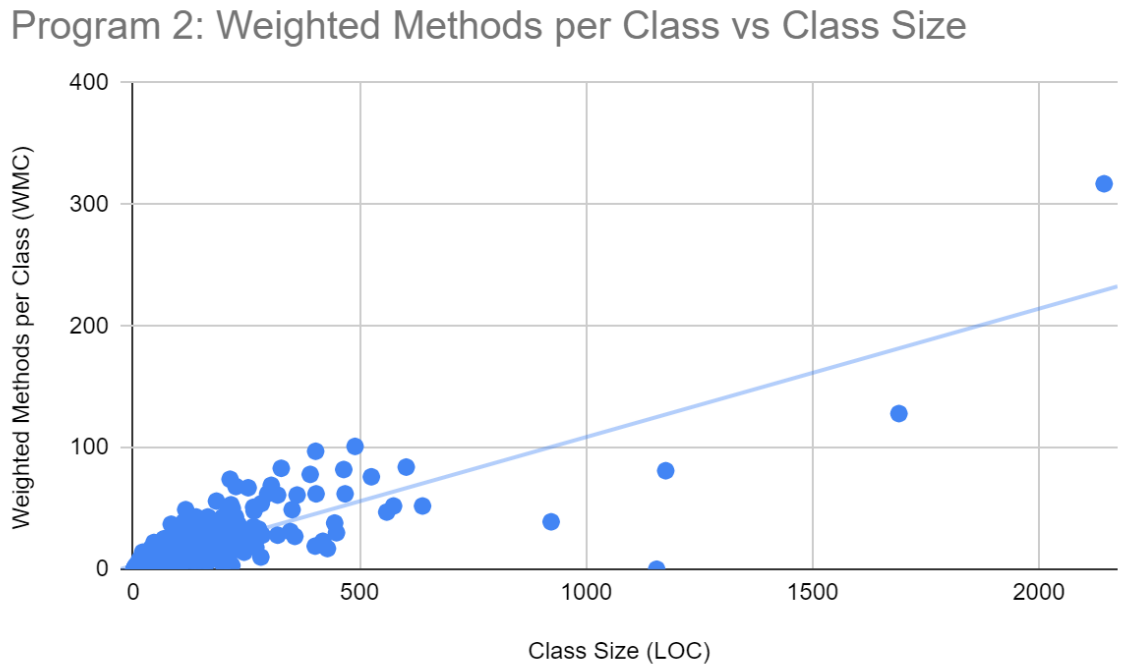


Graph 1: This first scatter plot graph displays the relation between WMC and CBO resulting in a tight clustering along a trend line with few outliers.

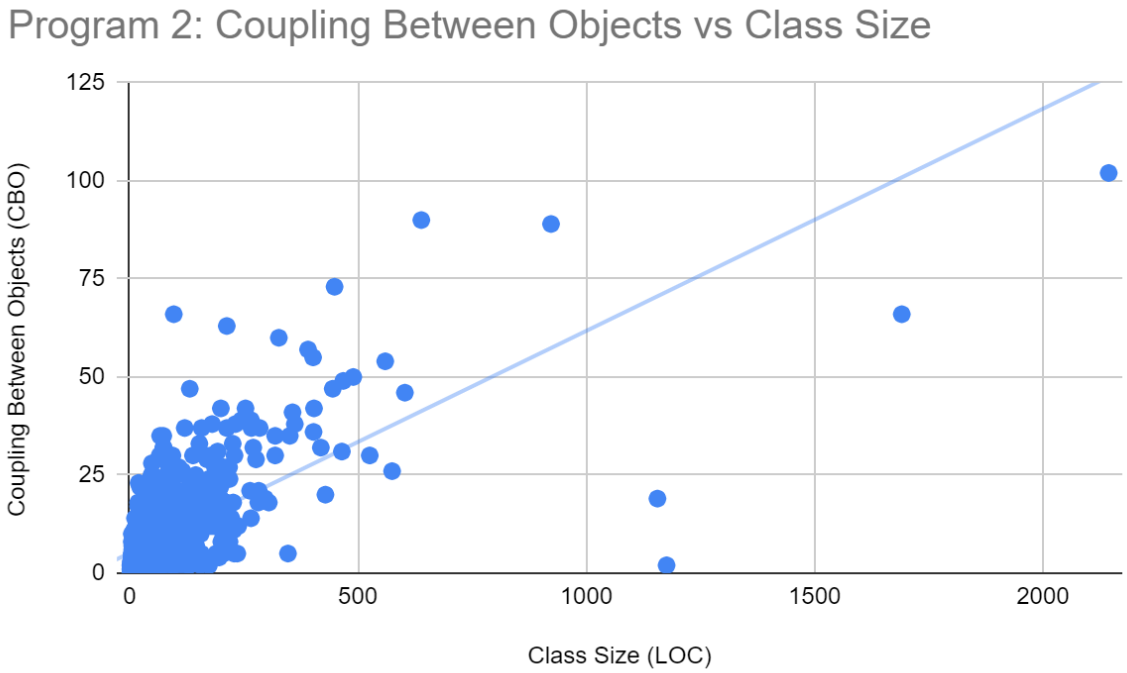


Graph 2: This scatter plot graph displays the relation between CBO and LOC resulting in a tight clustering with a number of outliers skewing the trend line to fall out of the main cluster of data.

Program #2: Mantis

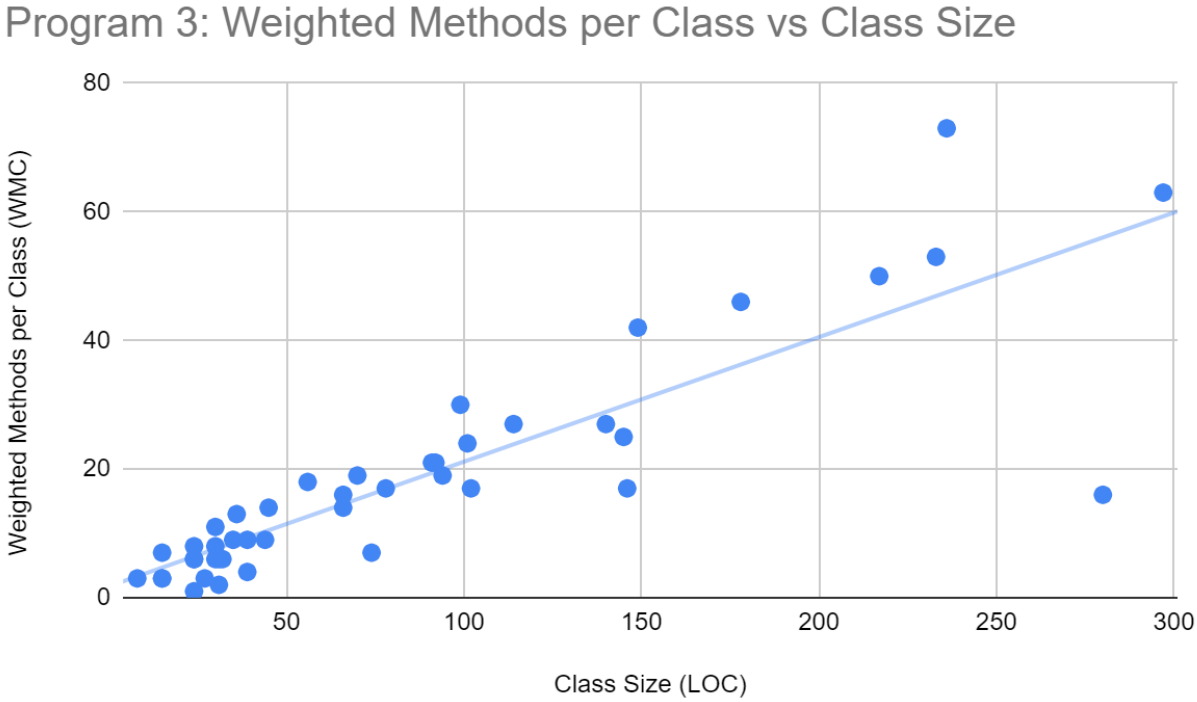


Graph 3:This scatter plot graph displays the relation between WMC and LOC resulting in a tight clustering along a trend line with few outliers.

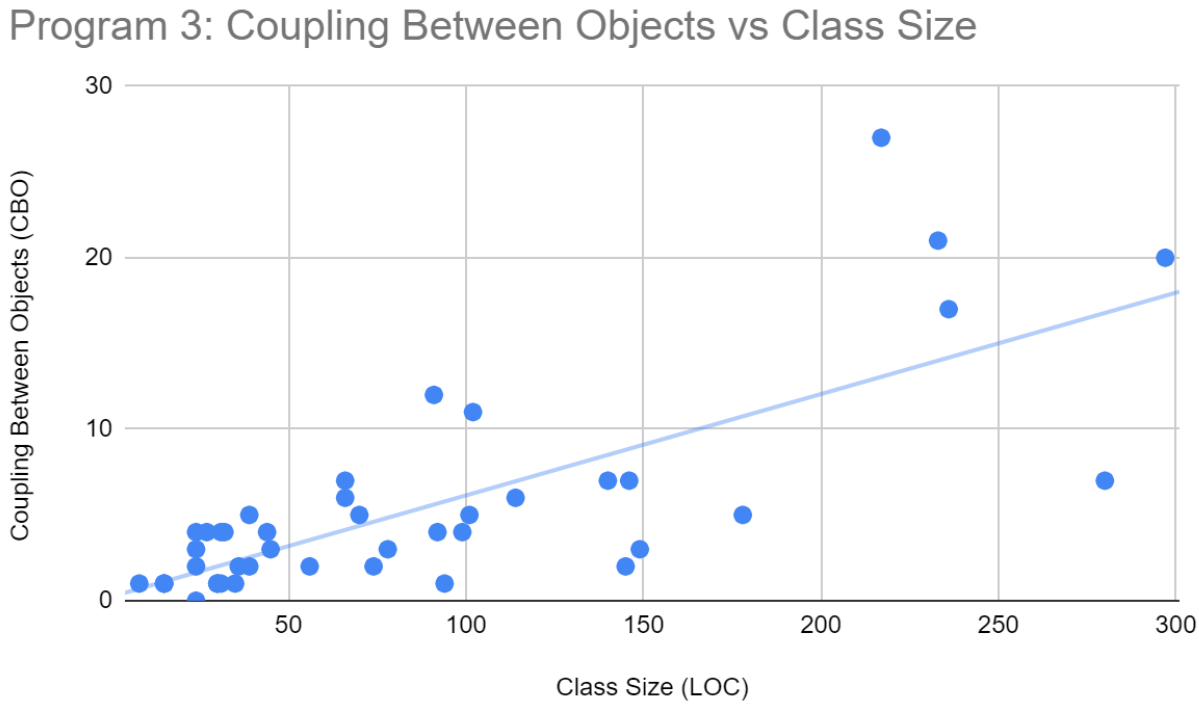


Graph 4: This scatter plot graph displays the relation between CBO and LOC resulting in a tight clustering with a number of outliers skewing the trend line to fall out of the main cluster of data.

Program 3: QtScrcpy

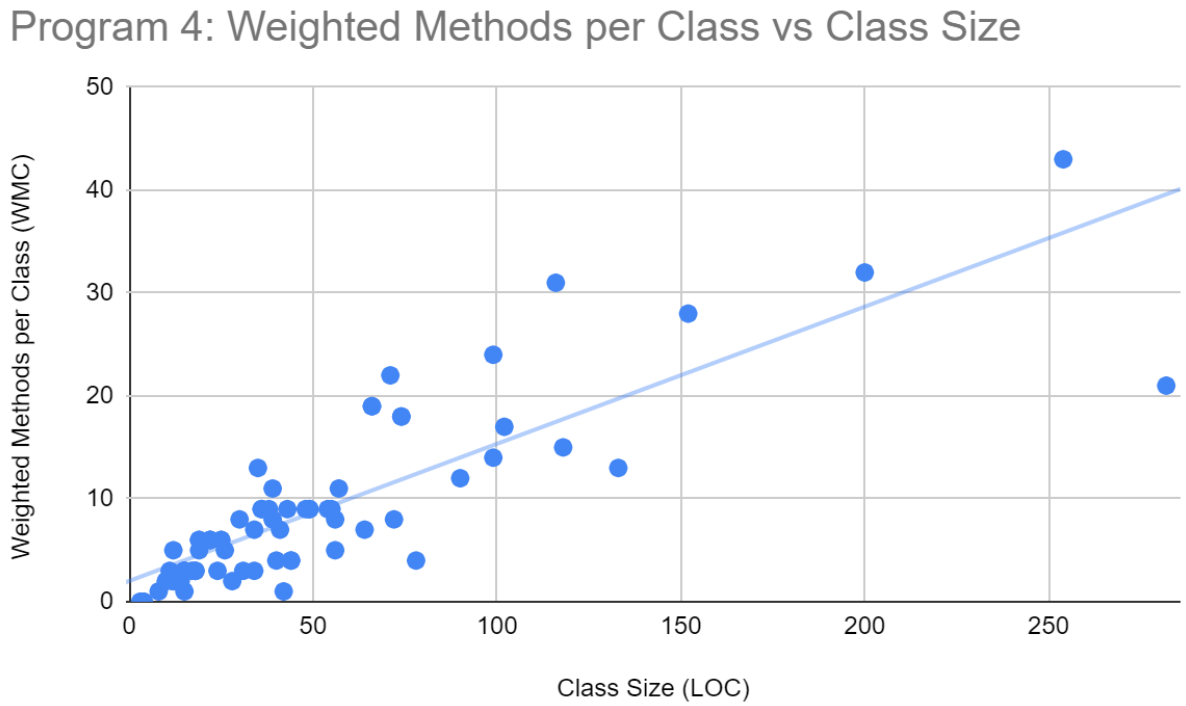


Graph 5: This scatter plot graph displays the relation between WMC and LOC resulting in a tight clustering along a trend line with few outliers.

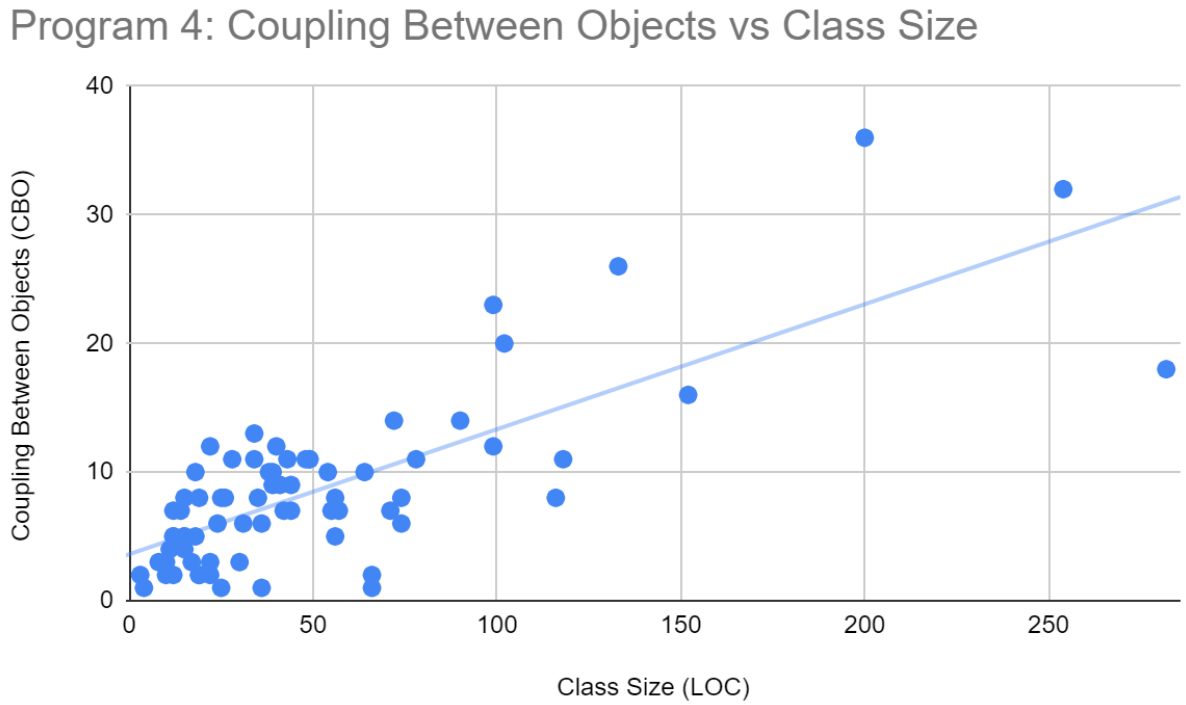


Graph 6: This scatter plot graph displays the relation between CBO and LOC resulting in a distributed clustering with a number of outliers; however the trend line can still be used to approximate forecasted values.

Program 4: Animation Samples

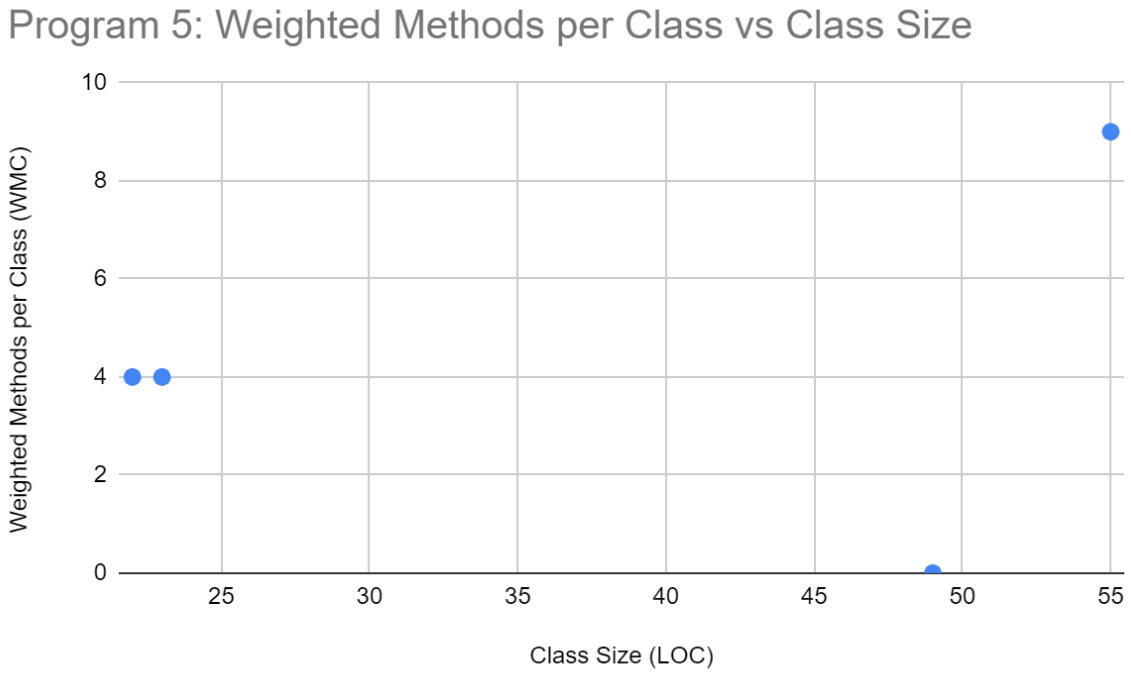


Graph 7: This scatter plot graph displays the relation between CBO and LOC resulting in a tight clustering with a number of outliers; however the trend line can still be used to approximate forecasted values.

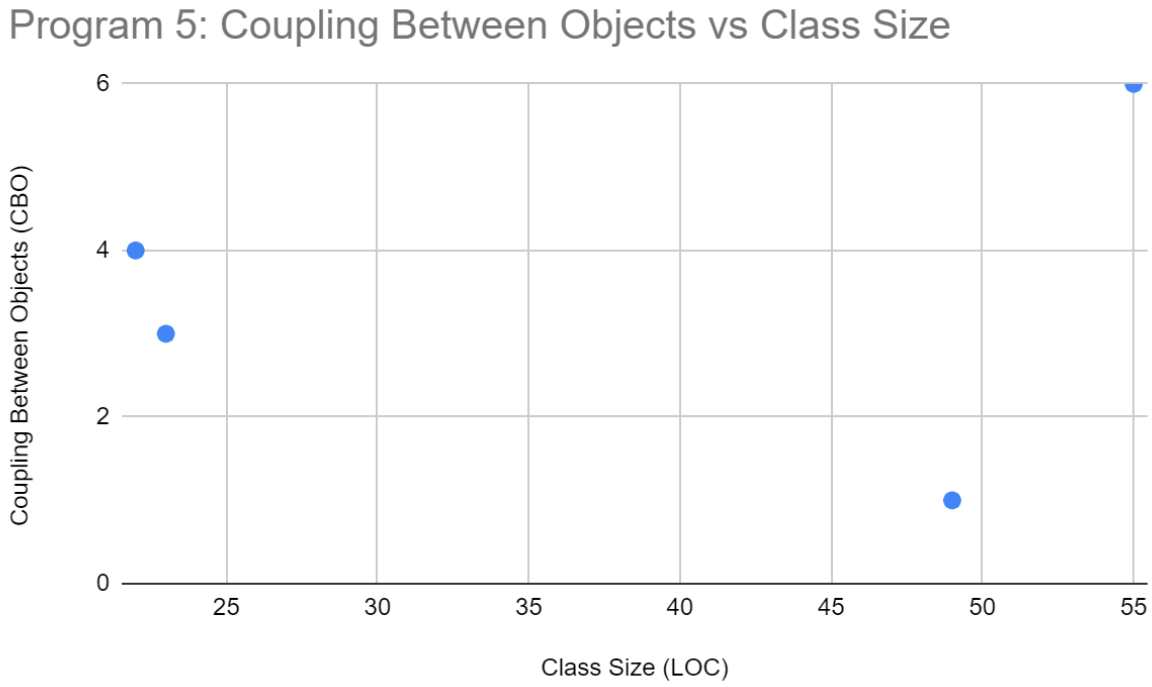


Graph 8: This scatter plot graph displays the relation between CBO and LOC resulting in a loose clustering with a number of outliers; however the trend line can still be used to approximate forecasted values.

Program 5: Booknotes



Graph 9: This scatter plot graph displays the relation between WMC and LOC with the few data points unable to give a reliable trend line.



Graph 10: This scatter plot graph displays the relation between CBO and LOC with the few data points unable to give a reliable trend line.

Section 5

Conclusions

Closing conclusions:

The finding that both CBO and WMC increased with the increase in class size in all five programs highlights the importance of careful software design and architecture. As the class size grows, the complexity of the code also increases, which can make it harder to maintain and modify over time. In particular, CBO and WMC are both metrics that have been shown to be related to maintainability issues, so this finding suggests that larger classes could be more difficult to maintain. This underscores the need for software developers to carefully consider the design of their code, particularly as the size of the codebase grows. By paying attention to metrics like CBO and WMC and striving to keep them at manageable levels, developers can help ensure that their code remains maintainable and easy to work with over the long term.

Next steps to determine the effect of class size on maintainability could involve further investigation into how class size affects other metrics, such as code complexity and code duplication. Additionally, it may be useful to explore how different development methodologies, such as agile or waterfall, affect the relationship between class size and maintainability. Another potential area of research could involve examining the impact of team size and collaboration on maintainability, as larger teams may have different communication and coordination challenges that could affect the maintainability of their code. Overall, while class size appears to have a significant impact on maintainability, additional measuring techniques are needed to fully understand the complex relationships between different factors that affect code quality and maintainability.

In conclusion, this paper presents a tool for measuring the impact of class size on software maintainability, by analyzing code smells related to complexity and coupling. The tool is based on the CK repository, which provides a collection of code smells, detection tools, and refactoring catalogs for several programming languages. The analysis of open-source Java projects from GitHub using the tool showed that large classes tend to have higher complexity and coupling than small classes, suggesting that class size has a negative impact on software maintainability. The results were supported by the trend lines in the scatter plot charts for the measurement results. However, there is a lack of reliable measurement for the Cohesion metric, which would have been a desirable metric for the objective. Additionally, it should be noted that the Coupling Between Objects metric may depend on the object being analyzed and is not necessarily nested underneath Lack of Cohesion of Methods in a difficulty hierarchy.

Citations:

Android Animation Samples: Google. (2019). Android animation samples (Version 1.0.0) [Computer software]. GitHub. https://github.com/android/animation-samples

Basili, V. R., Briand, L. C., & Melo, W. L. (1996). A validation of object-oriented design metrics as quality indicators. IEEE Transactions on software engineering, 22(10), 751-761.

Blender: Blender Foundation. (2022). Blender (Version 3.1) [Computer software]. https://www.blender.org/

Booknotes: Mihaylov, P. (2022). Booknotes (Version 1.0.0) [Computer software]. GitHub. https://github.com/preslavmihaylov/booknotes

Chidamber, S. R., & Kemerer, C. F. (1994). A metrics suite for object oriented design. IEEE Transactions on software engineering, 20(6), 476-493.

Fowler, M. (2018). Refactoring: Improving the Design of Existing Code (2nd ed.). Addison-Wesley Professional.

Python: Python Software Foundation. (2021). Python (Version 3.10.0) [Computer software]. https://www.python.org/

QtScrcpy: Ran, B. (2021). QtScrcpy (Version 2.4.0) [Computer software]. GitHub. https://github.com/barry-ran/QtScrcpy

Wedyan, F. (n.d.). Object Oriented Metrics [PowerPoint slides]. Lewis University.

Weisfeld, M. (2021). Object-Oriented Thought Process (5th ed.). Pearson Education.