**Empirical Study Report: Impact of Class Size on Software Maintainability**

***Section 1: Objectives, Questions, and Metrics***

**Objective:**

To analyze how the size of a class influences its maintainability in software systems.

**Research Question:**

What is the relationship between class size (measured in lines of code) and software maintainability?

**Metrics:**

We will utilize the following Chidamber and Kemerer (C&K) metrics to measure maintainability:

1. **Weighted Methods per Class (WMC):** Indicates the complexity of a class by summing the complexity of its methods. Higher WMC values suggest greater complexity and potentially lower maintainability.
2. **Depth of Inheritance Tree (DIT):** Measures the inheritance levels from the object hierarchy top. A deeper inheritance tree can complicate maintenance due to increased complexity.

**Class Size Metric:**

* **Lines of Code (LoC):** The primary metric for determining class size.

***Section 2: Subject Programs (Data Set)***

**Selection Criteria:**

* Programs must have at least 10,000 lines of code.
* Programs should be at least 3 years old.
* Programs must have contributions from at least 3 developers.

**Chosen Java Projects from GitHub:**

1. Apache Commons Lang: Utility library extending the java.lang package.
2. JUnit: Framework for unit testing in Java.
3. Apache Maven: Build automation tool for Java projects.
4. Spring Framework: Comprehensive framework for enterprise Java applications.
5. Google Guava: Core libraries for Java from Google.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Project Name** | **LoC** | **Age (Years)** | **Contributions** | **Description** |
| Apache Commons Lang | 26792 | 10 | 166 | Extends the functionality of java.lang package |
| Junit5 | 620906 | 7 | 174 | Framework for testing Java applications |
| Apache Maven | 52836 | 13 | 153 | Java build automation tool |
| Spring Framework | 175871 | 12 | 387 | Framework for enterprise Java applications |
| Google Guava | 388778 | 8 | 273 | Core libraries for Java |

***Section 3: Tool Description***

**Tool Utilized:**

We employed the CK tool for static code analysis to extract C&K metrics. This tool, available on GitHub, enables the measurement of various object-oriented metrics, including WMC and DIT.

**Tool Citation:**

* Aniche, Mauricio. "CK: A Tool for Calculating Chidamber and Kemerer Metrics." GitHub Repository, 2021. Available at: CK Tool GitHub.

***Section 4: Results***

The results were obtained for WMC and DIT metrics for each class within the selected projects, alongside their LoC. These findings are illustrated through bar charts and line graphs to identify trends and correlations.

Average Values of WMC, DIT, and LOC for the five projects

|  |  |  |  |
| --- | --- | --- | --- |
| **Project Name** | **Avg (WMC)** | **Avg (DIT)** | **Avg (LOC)** |
| Apache Commons Lang | 14.30 | 1.57 | 82.52 |
| Junit5 | 4.89 | 1.32 | 29.27 |
| Apache Maven | 10.39 | 1.44 | 46.93 |
| Spring Framework | 7.6 | 1.47 | 41.71 |
| Google Guava | 6.86 | 1.79 | 40.26 |

**Example Charts and Analysis:**

*Chart 1: WMC vs. Class Size (LoC)*

*Chart 2: DIT vs. Class Size (LoC)*

**Observations:**

* A positive correlation between WMC and LoC was observed; larger classes generally exhibit higher complexity.
* DIT did not exhibit a significant correlation with class size, suggesting that inheritance complexity is influenced by design choices rather than class size.

***Section 5: Conclusions***

This empirical study investigated the impact of class size on program maintainability by examining five well-known Java projects: Apache Commons Lang, JUnit, Apache Maven, Spring Framework, and Google Guava. Using the Chidamber and Kemerer (C&K) metrics, specifically Weighted Methods per Class (WMC) and Depth of Inheritance Tree (DIT), the study sought to identify connections between class size, as measured in Lines of Code (LoC), and these key maintainability metrics.

The investigation found a substantial positive link between class size and the WMC metric. Larger classes typically have higher WMC values, suggesting greater complexity. This increasing complexity might reduce maintainability, making it more difficult for developers to manage, comprehend, and adjust the code. This pattern was similar across all of the investigated projects, implying that as classes expand in number, so does their complexity, and hence the effort required to manage them.

However, there was no significant association between the DIT metric and class size. This shows that the complexity of inheritance depth is determined more by design decisions made throughout the program development process than by class size. The architectural choices involving inheritance hierarchies have a greater impact on DIT values, demonstrating that inheritance complexity derives more from the intentional design of the program architecture than the size of individual classes.

These findings lead to the conclusion that, while class size management is critical for controlling complexity and enhancing maintainability, designing the inheritance structure necessitates rigorous architectural planning. The study emphasizes the need of limiting class sizes to reduce complexity and improve maintainability. At the same time, it underlines the importance of strategic design decisions involving inheritance in determining a software system's maintainability.

To promote program maintainability, developers should limit class sizes to avoid excessive complexity. They should also focus on smart architectural planning, particularly in terms of inheritance structures, to ensure that the depth of inheritance does not overcomplicate the system. By balancing these factors, developers can build more maintainable and robust software systems.

Overall, the study emphasizes how important deliberate architectural design and class size management are to the maintainability of software. Software developers can benefit greatly from these insights by using them to design strategies that improve the quality and maintainability of their code. By carefully controlling class sizes and making thoughtful design decisions, developers can reduce complexity and create software systems that are simpler to update and expand over time.

***References:***

1. Chidamber, S.R., and Kemerer, C.F. "A Metrics Suite for Object-Oriented Design." IEEE Transactions on Software Engineering, vol. 20, no. 6, 1994, pp. 476-493.
2. CK Tool GitHub Repository. Available at: CK Tool GitHub.
3. Project repositories on GitHub: Apache Commons Lang, JUnit, Apache Maven, Spring Framework, Google Guava.

***Submission:***

The full report and raw data are available on GitHub at the following link: *https://github.com/harikamareedu7/Object-Oriented-Development.git*