Jade Provance

Project 1

First empirical study: Effect of class size on software maintainability

**Section 1**

Objective: To determine the effect of class size on software maintainability using the CK metrics.

Questions:

a) How does class size affect maintainability in the selected Java projects?

b) What is the relationship between class size and maintainability as measured by the CK metrics?

c) How do different CK metrics (WMC and CBO) compare in measuring the effect of class size on maintainability?

Metrics:

a) CK Metric: Weighted Methods per Class (WMC)

b) CK Metric: Coupling between Object classes (CBO)

c) Number of lines of code (LOC)

I analyzed programs that were written in Java, were at least 100,000 in size, at least 5 years old, had at least 20 developers, and came from a variety of domains. I selected projects that were written in Java because I used CK metrics, which are designed for object-oriented programming languages like Java. I chose ones that had at least 100,000 lines of code because larger projects tend to be more complex and harder to maintain which makes them good candidates for studying the effect of class size on maintainability. I chose ones that were older than 5 years old because they were likely to have gone through various maintenance tasks. Projects with at least 20 developers were chosen because they tend to have more complex code due to more people working on them which can also affect maintainability. Lastly, I picked projects that had come from a variety of domains because they can have different requirements for software maintainability.

**Section 2**

Programs chosen for study and their attributes.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Program Name | Age (years) | Size (LOC) | Number of Developers | Domain |
| Apache Flink | 11 | 420,000 | 125 | Distributed data processing and analysis |
| Jenkins | 19 | 151,000 | 742 | Continuous integration and delivery |
| Apache Atlas | 8 | 250,000 | 38 | Data governance and metadata management |
| Spring | 21 | 600,000 | 720 | Enterprise Java framework |
| Apache Tika | 16 | 326,000 | 85 | Content analysis and detection |

**Section 3**

I used the CK metrics tool to analyze the projects. It was created by Maurício Aniche and developed by a group of 23 developers using Java. It was designed to obtain the C&K metric values for measuring various software quality attributes, including maintainability. The tool is available on GitHub and can be downloaded from the following link: <https://github.com/mauricioaniche/ck>. The tool uses static analysis to calculate the metrics, and its output is provided in Excel sheets. One limitation of the tool is that it only works for Java programs so its results may not work with other programming languages. The tool was last updated in March 2023 and remains active.

CK reference

@manual{aniche-ck,

title={Java code metrics calculator (CK)},

author={Maurício Aniche},

year={2015},

note={Available in https://github.com/mauricioaniche/ck/}

}

**Section 4**

I chose to take the average WMC, LOC, and CBO values from each project so I could see if there was a direct correlation between those metrics and software maintainability.

Figure 1. The Average WMC vs Average LOC of Projects

Table 1. The average WMC and LOC values for each project

**Table

Description automatically generated**

Figure 2. The Average LOC vs Average CBO of Projects

Table 2. The Average LOC and CBO values for each project

**Table

Description automatically generated**

Figure 3. The Average WMC and CBO Values for Each Project

Table 3. The average WMC and CBO values for each project

**Table

Description automatically generated**

**Section 5**

The WMC and CBO metrics can give insight into how class size affects software maintainability.

Figure 1. displayed the average WMC and LOC values for each project. As the line of code increased so did WMC which is rational because as more lines are added there are more methods that are needed. High WMC and high LOC values could indicate that the software is more difficult to maintain. However, it’s natural for these two values to correlate.

In Figure 2. the LOC and CBO average values were shown. LOC can give insight into how class size affects software maintainability because if a class has a lot of code, it could be harder to understand and maintain. When there’s more code than it can become more difficult to implement changes without introducing bugs. The CBO measures the number of other classes that a class is coupled to. A higher value indicates that a class has a lot of dependencies which can make maintenance more difficult. All five projects had a relatively high level of code, but their CBO values were low. Based off the graph, the more lines of code there were the CBO values rose which is logical.

As stated earlier, a high WMC value indicates that a class has more methods, which can make it more difficult to test and maintain. Meanwhile, a low WMC value indicates that a class has fewer methods, which could make it easier to test and maintain. In Figure 3, the five projects WMC’s were quite close to each other. All of them averaged below 20. This can indicate that software maintainability is easier with fewer class sizes because there’s less to test and maintain. The CBO metric measures the number of other classes that a class is coupled to. High CBO values indicate that a class has many dependencies on other classes, which can make it more difficult to maintain because changes to one class can affect other classes. Also shown in Figure 3, all five averaged below 10 indicating another sign of easier maintenance.

Based off the five projects, the average WMC and CBO values were relatively low compared to the lines of code. It’s uncertain whether class size has effect on software maintainability without studying more programs. This is because the WMC and CBO values were low. This could indicate that the programs have many duplicated code segments which can make it harder to maintain because changes to one segment may need to be replicated in several places. However, since the complexity and coupling of the class may not be that high, it may still be relatively easy to maintain.

**References**

Apache Flink: <https://github.com/apache/flink>

Jenkins: <https://github.com/jenkinsci/jenkins>

Apache atlas: https://github.com/apache/atlas

Spring Framework: <https://github.com/spring-projects/spring-framework>

Apacha tika: https://github.com/apache/tika