Impact of Class Size on Software Maintainability Using C&K Metrics

Bhanuprakash Banda and Venkata Krishna Vamsi Polagana

Lewis University

CPSC-60000; Object Oriented Development

Dr. Ziad Al-Sharif

05/26/2024

Contents

Section 1: Objectives, Questions, and Metrics (GQM Approach)

Objective:

To investigate the impact of class size on the maintainability of software components.

Questions:

1. How does the size of a class affect the maintainability of software components?
2. Is there a correlation between larger class sizes and lower levels of maintainability?
3. What specific aspects of maintainability, such as complexity, cohesion, and coupling, are influenced by variations in class size?

Metrics:

1. Weighted Methods per Class (WMC): The Weighted Methods per Class (WMC) metric quantifies the complexity of a class by calculating the total number of methods in the class. Greater values of Weighted Methods per Class (WMC) indicate more complexity, which might have a detrimental impact on maintainability.
2. Lack of Cohesion in Methods (LCOM): LCOM quantifies the absence of coherence among the methods within a class. More LCOM values indicate inferior organization and potential difficulties in maintaining the code, as classes with more cohesion are typically more understandable and adaptable.
3. Depth of Inheritance Tree (DIT): The Depth of Inheritance Tree (DIT) measures the number of levels in a class's inheritance hierarchy. A higher DIT number indicates

increased complexity and probable challenges in maintaining and comprehending the class because of its location in the inheritance hierarchy.

4. Coupling Between Objects (CBO): The CBO metric quantifies the degree of interdependence between a class and other classes within the system. More significant interdependencies are shown by higher CBO values. This can lead to more complexity and less maintainability since changes in one class may require changes in related classes.

Section 2: Description of Subject Programs (Dataset)

Criteria:

- Minimum size: 25000
- Minimum age: 6 years
- Minimum number of contributors: 150

Justification:

When a project has a minimum size requirement of 25,000 lines of code, larger projects generally have more intricate codebases, which offer a more comprehensive dataset for analysis. Enforcing a minimum size guarantee that the projects are of sufficient magnitude to incorporate a wide range of code patterns and architectural styles. Through the analysis of larger projects, we may do a more thorough investigation of software maintainability difficulties, as these projects usually involve a broader range of complex challenges and subtleties.

We selected the minimum age requirement of six years based on the observation that older projects typically undergo several iterations, bug patches, and upgrades, demonstrating real-world maintenance difficulties. The durability of a project indicates its ongoing significance and diligent upkeep, making it more appropriate for examining long-term trends in maintainability. Furthermore, older projects may have acquired technical debt over time, making it interesting for evaluating the effects of aging on software maintainability.

Finally, the requirement of having a minimum of 150 contributors ensures that the chosen projects have significant community participation and a wide range of perspectives in the code's development. Projects that involve numerous contributors often result in a broader spectrum of

coding styles, techniques, and norms, which enhances the dataset with diverse coding patterns. Moreover, projects that involve many participants are more likely to have undergone thorough peer reviews and collaborations, providing valuable knowledge about collaborative software development processes. This criterion assists in the selection of projects that demonstrate genuine, extensive, and actively updated software systems in the real world.

Table:

Project Name	**Lines of Code**	**Age (Years)**	**Number of Contributors**	**Description**
Arthas	28963	6	150	Alibaba specifically designed Arthas, an open-source Java diagnostic tool, to address production faults in Java applications without requiring code modification or server restarts. It assists developers in detecting issues in production scenarios where conventional debugging techniques are not feasible. The main features comprise verifying class loading, decompiling classes, examining classloader statistics, monitoring method invocations and system metrics, and additional functionalities. Arthas provides support for multiple systems, interactive command-line

				modes, telnet, and web interfaces for both local and remote diagnostics.
Nacos	45429	6	237	Nacos is a platform specifically created to facilitate the process of dynamically discovering services and managing configurations. It facilitates the registration and discovery of services in a convenient manner. Nacos provides support for a wide range of services and offers real-time health checks to ensure their dependability. Moreover, Nacos offers dynamic configuration management, eliminating the need for redeployment upon configuration modifications. Additionally, it offers support for dynamic DNS services and includes a dashboard for efficiently managing service metadata, configuration, health, and metrics statistics. In general, Nacos streamlines the process of constructing cloud-native applications and microservices platforms.
Java design patterns	28281	10	268	The "Design Patterns Implemented in Java" project demonstrates a range of design patterns used to address typical programming issues in

				the Java language. The project offers meticulously documented source code examples and lessons for each design, created by skilled programmers from the open-source community. These patterns enhance the legibility of code, expedite the development process, and mitigate the occurrence of nuanced problems. The project prioritizes simplicity, according to ideas such as KISS (Keep It Simple, Stupid) and YAGNI (You Aren't Gonna Need It). Users have the ability to explore patterns based on overarching descriptions, source code, tags, or categories such as creational and behavioral. The objective is to provide pragmatic, object-oriented resolutions for developers.
Apache Dubbo	39817	8	393	Apache Dubbo is a Java-based open-source RPC (Remote Procedure Call) framework that is known for its high-performance capabilities. The system facilitates transparent interface-based remote procedure calls (RPC), intelligent distribution of workload, and automatic registration and identification of

				services. Dubbo provides a high level of flexibility, the ability to route traffic at runtime, and a graphical system for managing services. To obtain additional information, please refer to the official website, which provides guidelines, documents, and updates.
Netty	79582	14	362	Netty is a framework specifically developed for creating and managing network applications. It is built to be efficient, reliable, and easy to maintain, making it ideal for developing protocol servers and clients that require high performance. It facilitates streamlined networking operations and can be constructed utilizing the most up-to-date stable versions of OpenJDK and Apache Maven. Netty is well-suited for the development of network applications that can handle large amounts of data and process it efficiently.

Section 3: Tool Description

CK is an advanced Java code metrics gathering tool that uses static analysis to produce a range of metrics at the class-level and method-level. It does not require compiled code. CK provides developers with a diverse set of metrics, enabling them to have a thorough comprehension of their code's architecture, intricacy, and ease of maintenance. The tool emphasizes various metrics, such as Coupling Between Objects (CBO), Depth of Inheritance Tree (DIT), Number of Children (NOC), and Lines of Code (LOC). CK is an excellent resource for reviewing and improving code quality in Java projects due to its comprehensive range of metrics.

An important characteristic of CK is its capacity to assess the level of coupling between items, often known as CBO (Coupling Between items). This statistic quantifies the quantity of dependencies that a class has on other classes, offering a clear understanding of the level of interdependence within the codebase. A revised iteration of the CBO metric considers both the dependencies that a class has on other classes and the dependencies that other classes have on it, providing a more nuanced understanding of coupling. Another crucial indicator is the Fan-In and Fan-Out, which quantify the number of classes that refer to a certain class and the number of other classes that a specific class refers to, respectively. These metrics aid in comprehending the progression of control and interdependencies within the system, which is essential for upholding and restructuring intricate codebases.

CK also incorporates metrics for quantifying the complexity of inheritance and class hierarchy, such as Depth of Inheritance Tree (DIT) and Number of Children (NOC). The Depth of Inheritance Tree (DIT) is a metric that quantifies the number of ancestor classes a given class has. This metric is useful for assessing the level of inheritance and the potential complexity involved in

comprehending and managing the code. The NOC, however, quantifies the direct subclasses of a class, demonstrating the possible influence of modifications in a superclass on its subclasses. These metrics are crucial for evaluating the design quality and maintainability of object-oriented systems.

CK also places significant emphasis on method-level metrics, such as the quantity of methods, visibility of methods, number of static invocations, and Response for a Class (RFC). These measurements offer comprehensive insights into the behavior and intricacy of specific approaches. For instance, the Weight Method Class (WMC) or McCabe's complexity metric quantifies the quantity of branch instructions in a class, emphasizing the possible complexity and testing effort needed for the class. The Lines of Code (LOC) metric, which quantifies the number of lines of code in a codebase, excludes empty lines and comments, thereby offering a precise assessment of the codebase's size.

CK also calculates different cohesion metrics, including Lack of Cohesion of Methods (LCOM) and its modified counterparts LCOM* and LCOM-HS. These metrics evaluate the level of interdependence between methods in a class, where higher cohesiveness typically signifies superior design and ease of maintenance. The Tight Class Cohesion (TCC) and Loose Class Cohesion (LCC) metrics enhance this evaluation by taking into account the direct and indirect relationships between methods, respectively. These cohesion metrics assist developers in identifying inadequately constructed classes that could potentially be enhanced through reworking to enhance modularity and readability.

The CK architecture is intentionally built to be modular and extendable, making use of existing design patterns like the Visitor, Notifier, and Factory patterns. The CKVisitor class, which inherits from the ASTVisitor class in the Eclipse JDT package, allows for thorough analysis and collection of metrics directly from the Abstract Syntax Tree (AST) of Java source code. CK can navigate

through different AST nodes in this approach, doing particular actions at each node to compute metrics. By utilizing the Notifier pattern via CKNotifier, CK is able to disseminate results to all registered observers, hence enabling a loosely linked architecture. The MetricsFinder class utilizes the Factory pattern to dynamically detect and create instances of metric collector classes, guaranteeing that CK can effortlessly adjust to new measurements without necessitating core alterations.

Section 4: Results

Bar Chart:

The purpose of this analysis is to investigate the correlation between different software maintainability measures and the number of classes in a collection of Java projects. Our primary focus is to investigate the impact of class size, as measured by lines of code (LOC), on its maintainability. To accomplish this, we will employ four essential measures from the Chidamber and Kemerer (C&K) suite:

1. The Weighted Methods per Class (WMC) metric quantifies the difficulty of a class by tallying the number of methods and assigning them weights based on their level of complexity.
2. Lack of Cohesion in Methods (LCOM) is a metric that measures the degree of cohesion among methods inside a class. It provides an indication of the level of interrelatedness between the methods in a class.
3. The Depth of Inheritance Tree (DIT) is a metric that quantifies the number of levels in a class's inheritance hierarchy. It serves as an indicator of the class's complexity and the potential issues that may arise in maintaining it.
4. Coupling Between Objects (CBO) is a metric that quantifies the level of interdependence between a class and other classes. Higher values of CBO indicate greater coupling and potential difficulties in maintaining the code.

To visually represent and examine these measurements, we will create a sequence of bar charts and scatter plots:

1. Bar charts will visually represent the numerical values of WMC, LCOM, DIT, and CBO for each class. Bar charts facilitate the straightforward comparison of measurements among several groups and the detection of any notable outliers.
2. A scatter plot will display the correlation between class size (LOC) and the maintainability metrics (WMC, LCOM, and CBO). By graphing these factors in relation to one other, we may analyze patterns and potential connections, which will aid in determining if there is a relationship between greater class sizes and increased complexity and decreased maintainability.

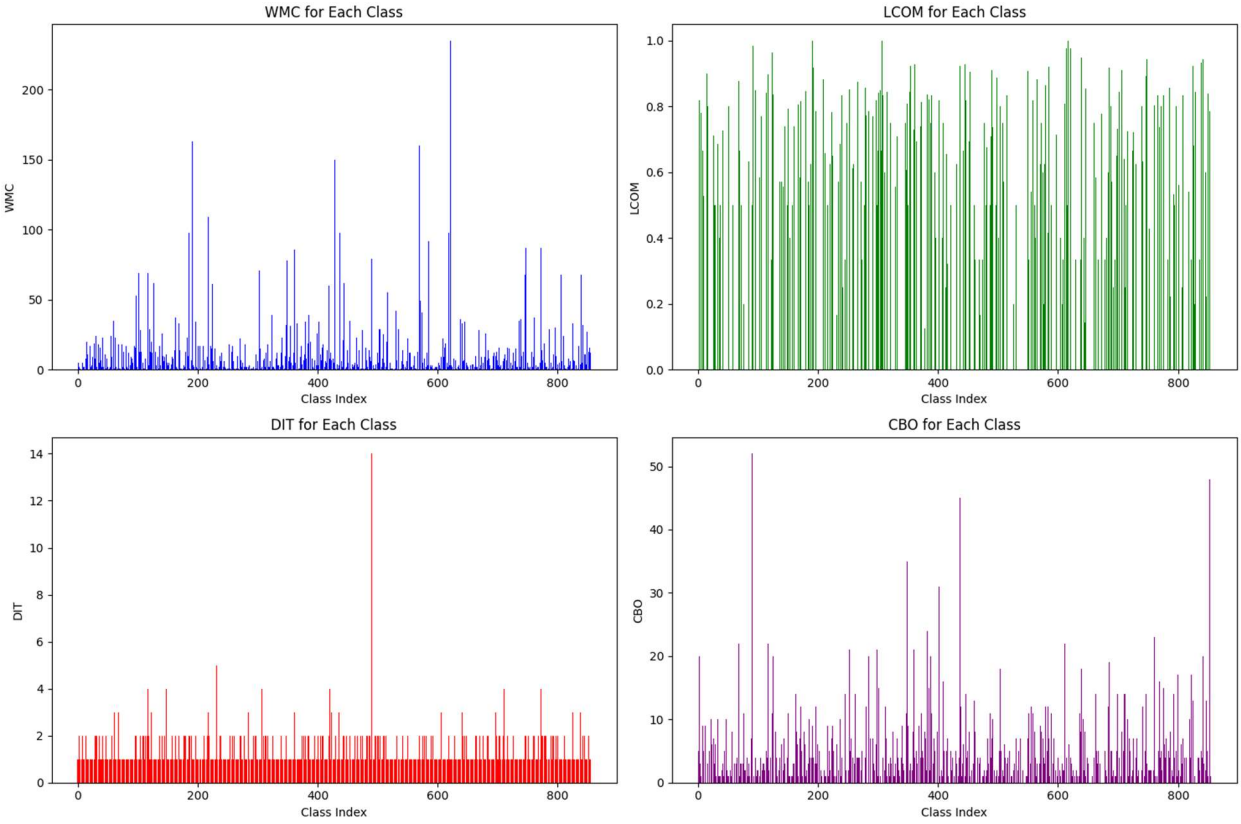

Figure 1: Bar charts of project Arthas

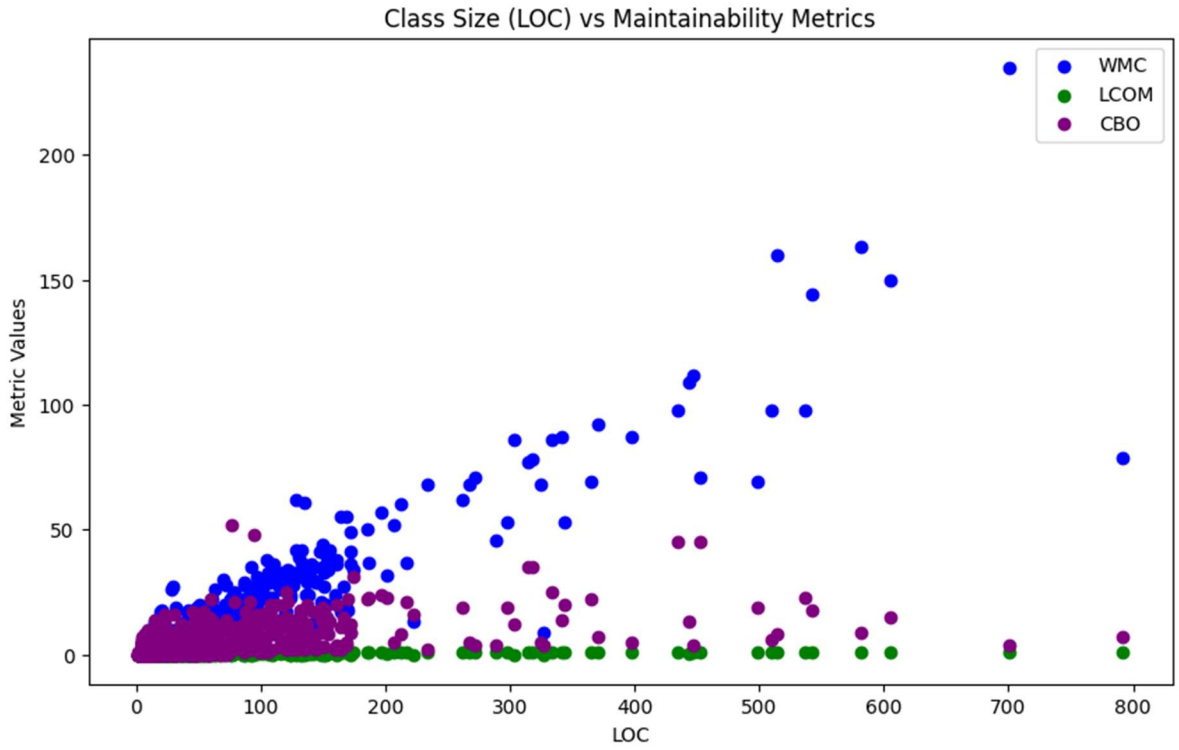

Figure 2: Scatter plot of project Arthas

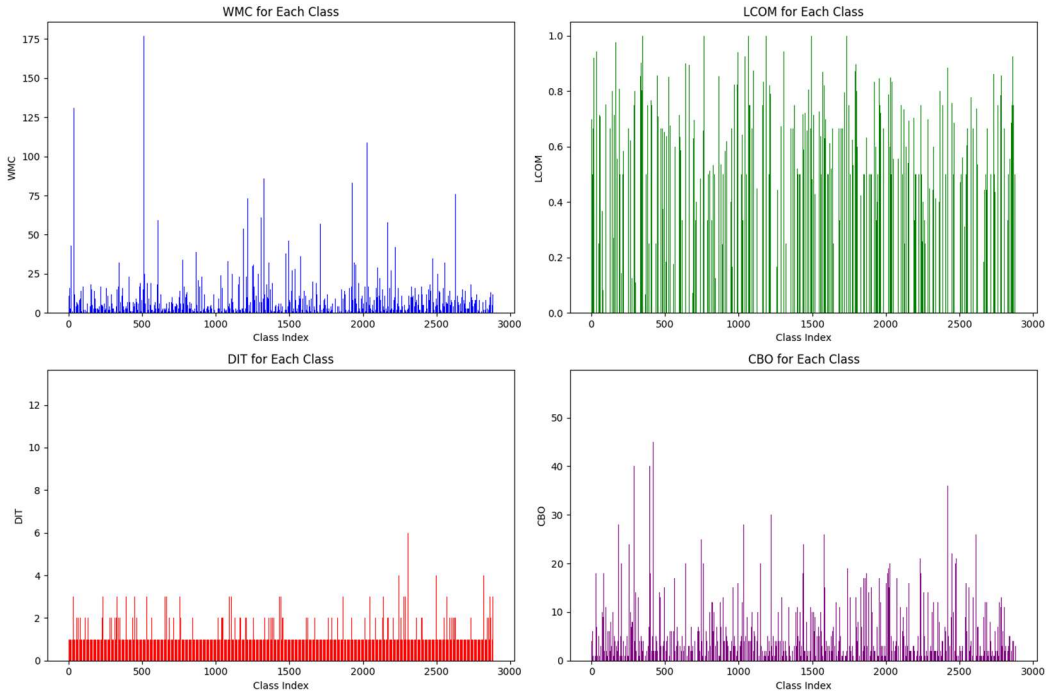

Figure 3: Bar chart of project Nacos

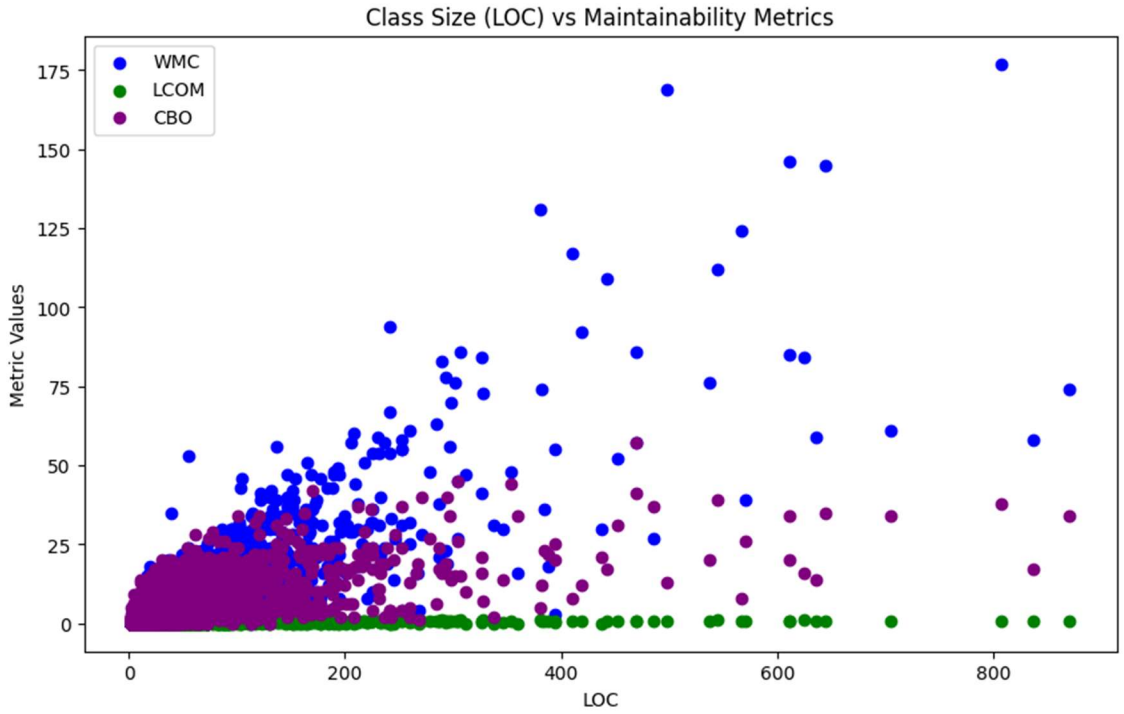

Figure 4: Scatter plot of project Nacos

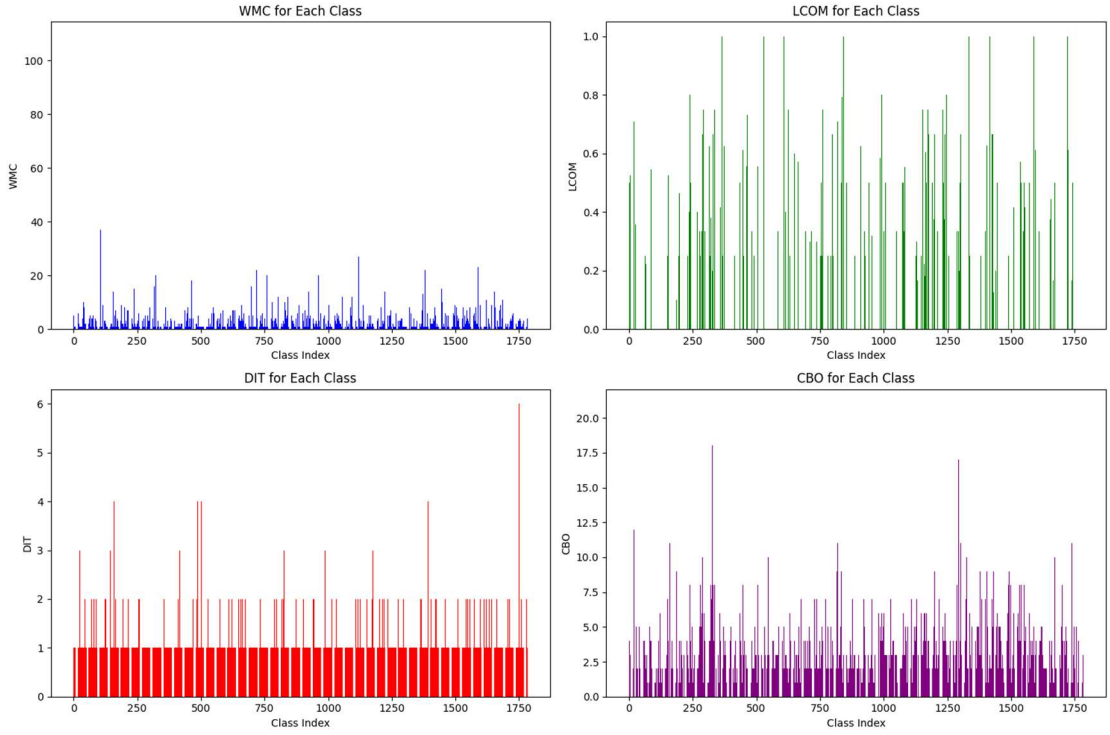

Figure 5: Bar chart of project Java Design Patterns

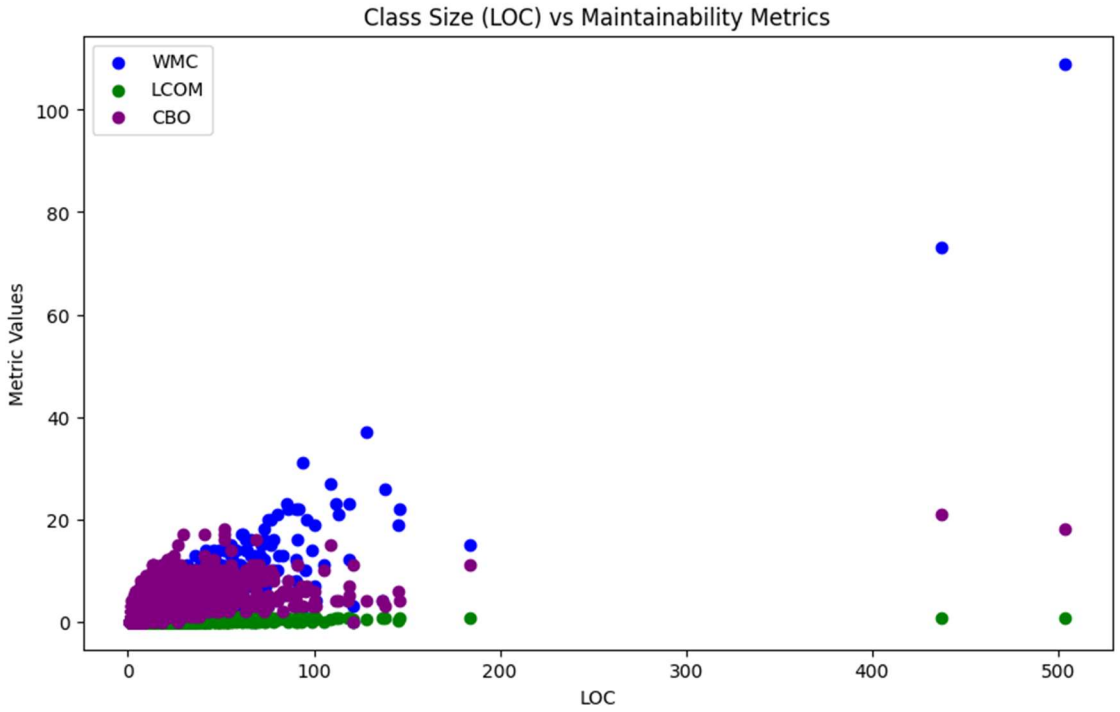

Figure 6: Scatter plot of project Java Design Patterns

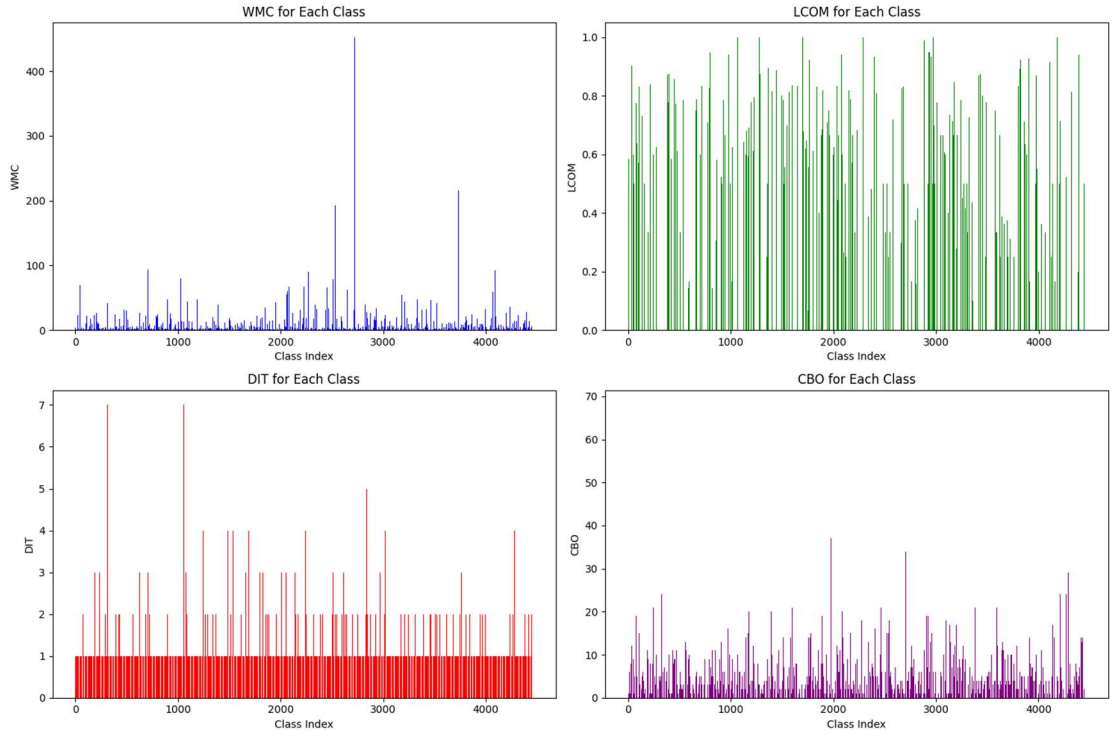

Figure 7: Bar chart of project Dubbo

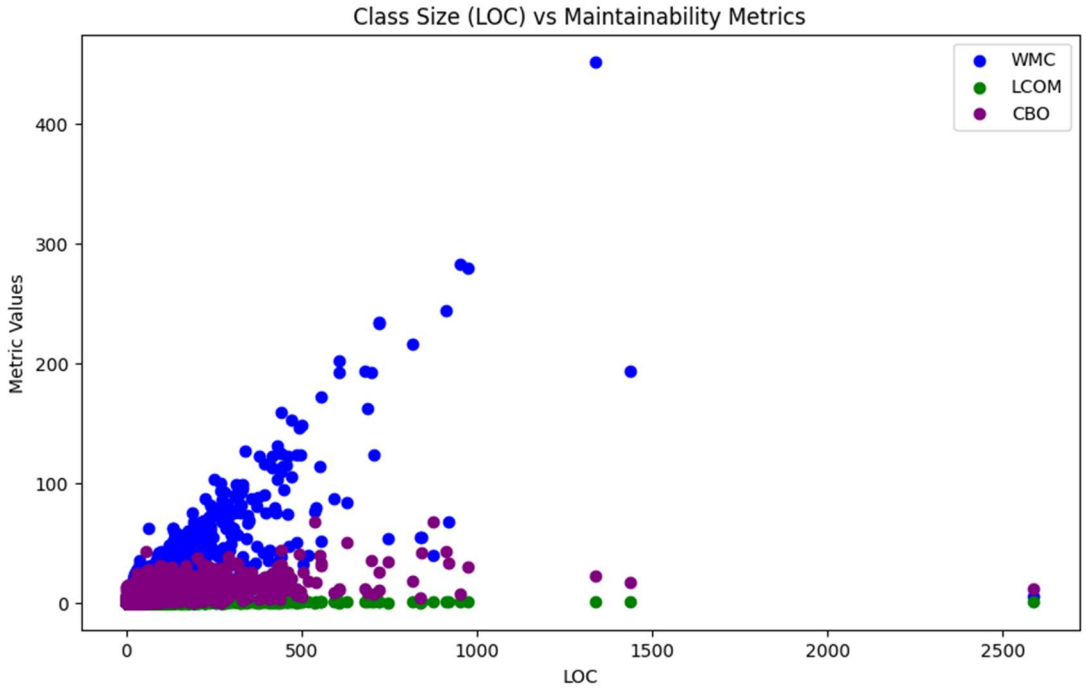

Figure 8: Scatter plot of project Dubbo

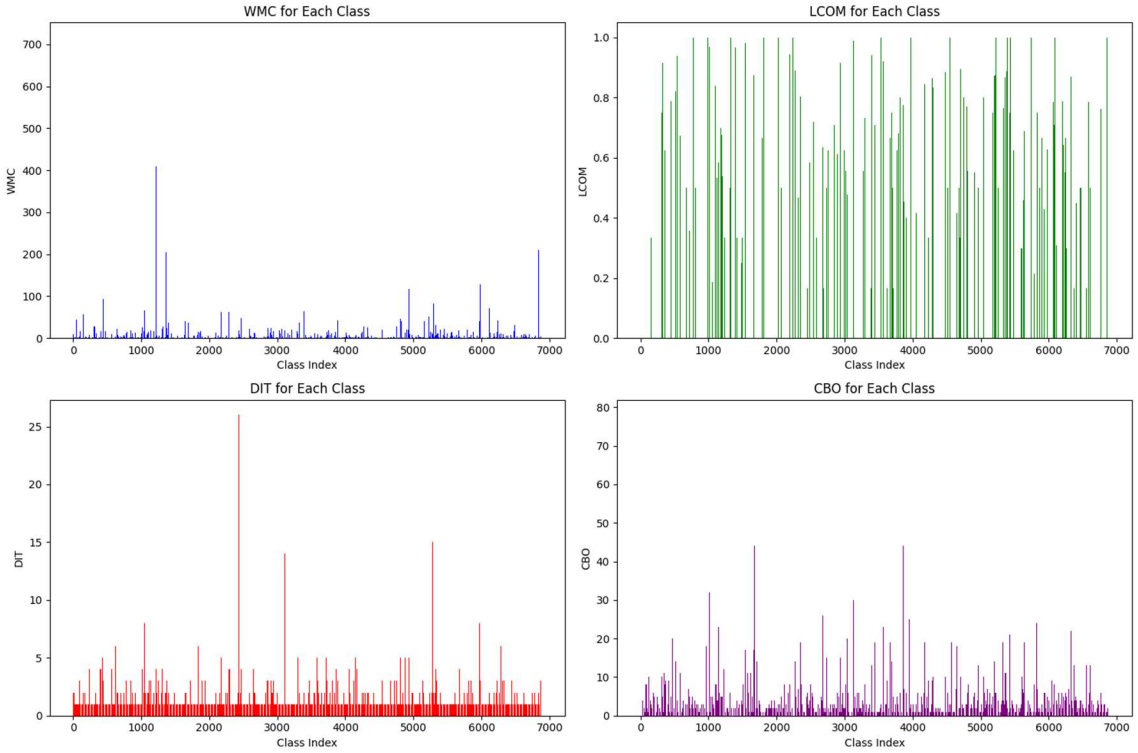

Figure 9: Bar chart of project Netty

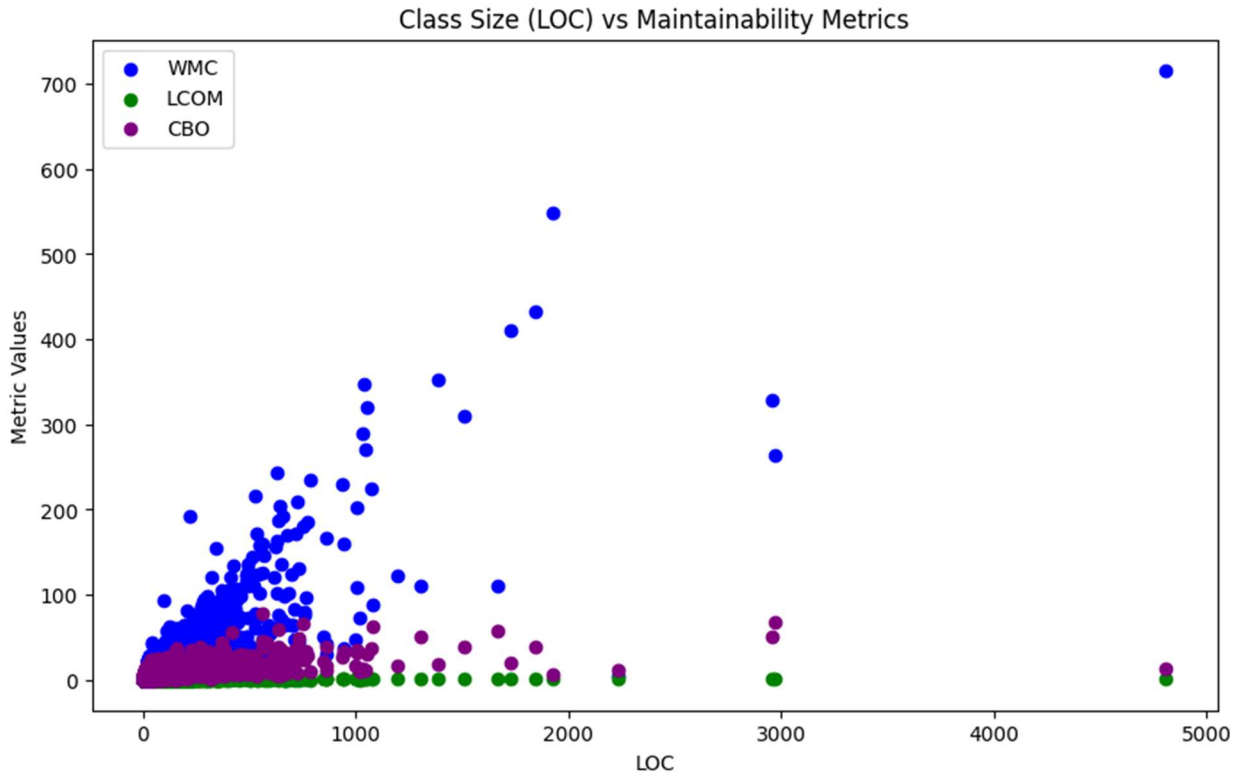

Figure 10: Scatter plot of project Netty

Correlation Matrix:

The requirement to investigate the intricate interconnections and interdependencies among different components within a dataset supports the decision to use a correlation matrix instead of line and bar charts. Contrary to line and bar charts, which usually display one or two variables, a correlation matrix allows us to analyze the pairwise connections between all variables at once. This comprehensive approach enables a more profound comprehension of how alterations in one variable can impact others, revealing patterns and connections that may not be evident when examining variables separately.

We consider multiple elements when generating a correlation matrix to produce a thorough and significant analysis. We primarily incorporate a wide array of variables pertinent to the

phenomenon under study. These variables can include many elements of the system or process under investigation, such as input parameters, performance indicators, or environmental conditions. By considering a diverse range of variables, we can better understand the system's complexity and determine possible factors that influence or forecast the observed outcomes.

Moreover, well-defined guidelines or thresholds that reflect the magnitude and direction of the relationships determine the understanding of correlation coefficients in the matrix. Positive correlations show a tendency for variables to move in the same direction, whereas negative correlations suggest an inverse relationship. The correlation coefficient's magnitude indicates the degree of association, with values closer to 1 or -1 suggesting greater associations.

The metrics contained in the correlation matrix offer significant insights into several facets of software maintainability, complexity, and design quality.

Lines of Code (LOC) is a metric that quantifies the total number of lines in a class or module, providing an indication of its size. The presence of LOC (Lines of Code) is critical because it has a significant impact on software maintainability and complexity, particularly in relation to class size. Similarly, the Total Number of Methods (TotalMethodsQty) represents the number of methods or functions within a class, which has an impact on its complexity and maintainability.

In addition, the Total Number of fields (totalFieldsQty) represents the count of attributes within a class, which impacts its overall size and complexity. Both metrics, totalMethodsQty and totalFieldsQty, are critical for understanding a class or module's structural complexity and maintainability.

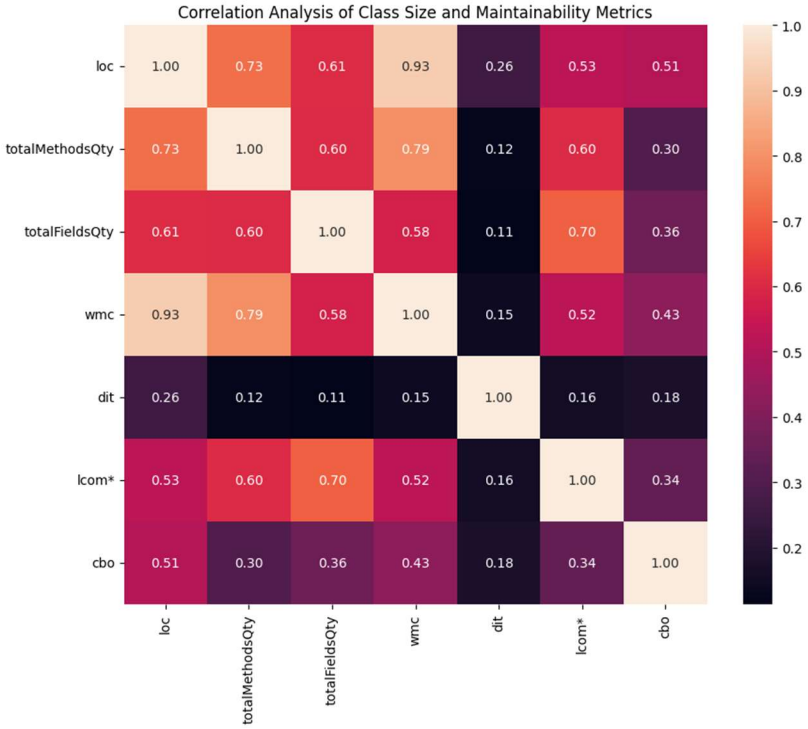

Figure11: Correlation matrix of project Arthas

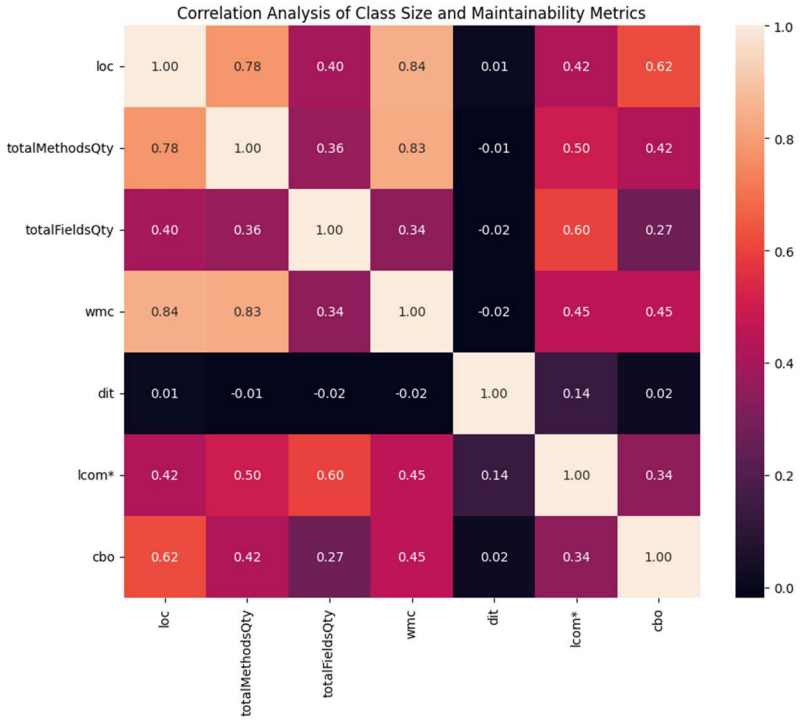

Figure 12: Correlation matrix of project Nacos

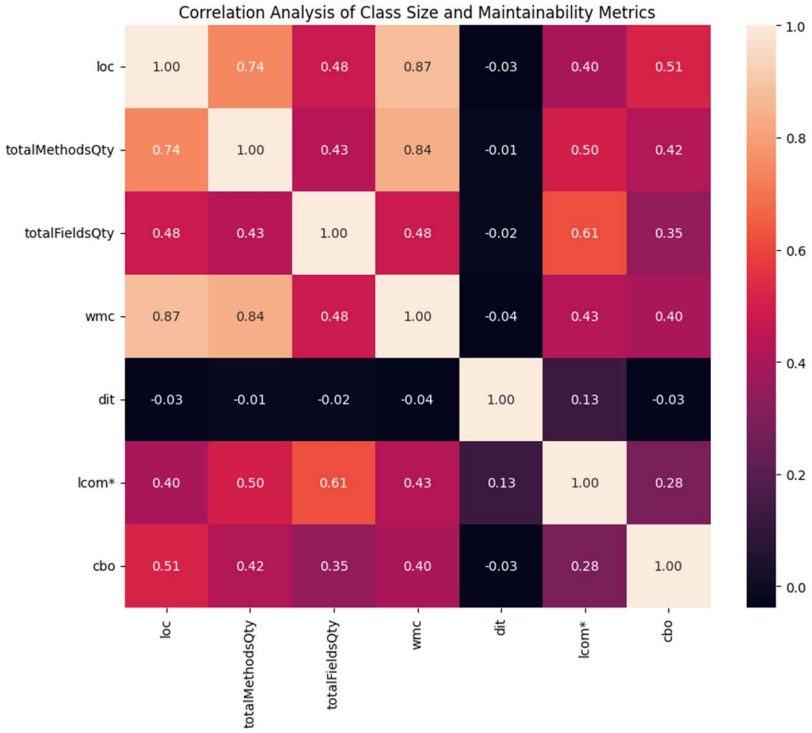

Figure 13: Correlation matrix of project Java Design Patterns

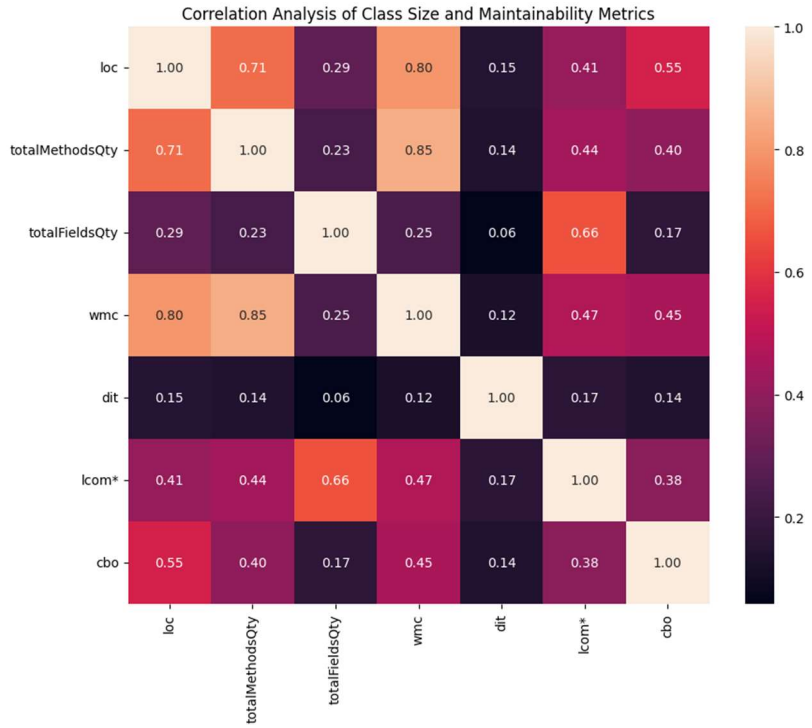

Figure 14: Correlation matrix of project Apache Dubbo

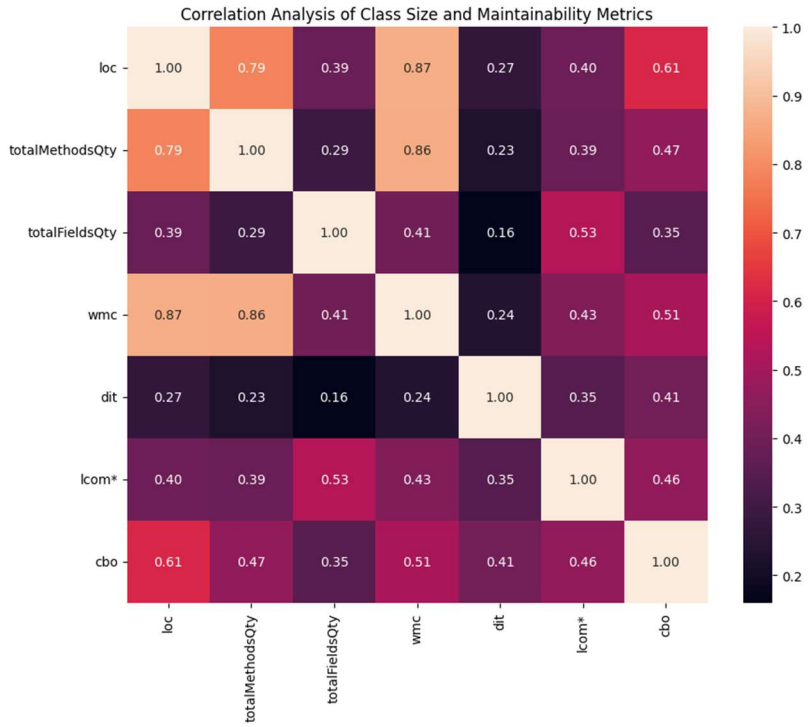

Figure 15: Correlation matrix of project Netty

Section 5: Analysis and Conclusions

Insights and Implications of Project Arthas

Based on Bar Chart and Scatter Plot:

The WMC values are generally low for most classes but spike significantly in several instances. High WMC values suggest these classes are more complex, potentially indicating lower maintainability. Such spikes in WMC could reflect classes with extensive responsibilities or many methods, which might complicate maintenance and understanding.

The LCOM values are spread across a wide range, with many classes showing mid to high levels of LCOM. Higher LCOM values indicate poor cohesion within the class, which can decrease maintainability by making the class harder to modify and understand.

Most classes have a low DIT value, suggesting shallow inheritance hierarchies. This is generally favorable for maintainability as it implies less complexity and fewer dependencies from inheritance. The few classes with higher DIT values could indicate deeper hierarchies, potentially increasing the complexity and making those classes more difficult to maintain due to inherited behaviors.

CBO values vary, with several peaks showing higher coupling. High coupling can reduce maintainability as changes in one class may affect multiple other classes, increasing the risk of bugs during modifications.

The scatter plot shows a general trend where larger class sizes correlate with higher WMC values, suggesting that larger classes are more complex. LCOM points are dispersed, indicating that class size does not consistently predict cohesion within a class. CBO also appears scattered but shows

some concentration in classes with larger sizes having higher coupling, which aligns with the understanding that larger classes might interact with more classes.

Based on Correlation Matrix:

There is a highly positive correlation between "loc" (Lines of Code) and "wmc" (Weighted Methods per Class) with a correlation coefficient of 0.93. Additionally, there is a strong positive correlation between "total method quantity" (total Number of Methods) and "wmc" with a correlation coefficient of 0.79. According to the data, there is a positive correlation between the number of lines of code and methods within a class and its difficulty, as measured by WMC. These findings indicate that classes with greater lines of code (LOC) and a higher number of methods tend to exhibit greater complexity, resulting in increased difficulty in comprehending, maintaining, and modifying them.

Modest positive correlations further emphasize other connections. For example, the variable "loc" exhibits modest correlations with "totalMethodsQty" (0.73) and "totalFieldsQty" (0.61), indicating that classes with a higher number of lines of code generally have a greater number of methods and fields, which adds to their overall complexity. The metric "Lack of Cohesion in Methods" (LCOM*) shows moderate positive associations with "Lines of Code" (LOC) (0.53) and "Total Methods Quantity" (totalMethodsQty) (0.60). This suggests that classes with a higher LOC and more methods may have lower cohesion within their methods, which can negatively impact maintainability. Furthermore, the metric known as "cbo" (Coupling Between Objects) exhibits moderate correlations with "loc" (0.62) and "totalMethodsQty" (0.42). This suggests that classes that are larger in size and have a greater number of methods are likely to have a stronger connection with other classes, resulting in increased complexity and decreased maintainability.

There are weak correlations between metrics such as "totalFieldsQty" and "dit" (Depth of Inheritance Tree), suggesting that these characteristics have a limited direct impact on class size or maintainability in this scenario.

Insights and Implications of Project Nacos

Based on Bar Chart and Scatter Plot:

The chart reveals several classes with spikes in WMC values. These peaks are indicative of classes with high complexity due to many methods or methods of high complexity, which could hinder maintainability. The distribution also suggests that while most classes maintain a lower complexity, those with high complexity are outliers that might require refactoring to improve maintainability.

The LCOM values are generally higher across many classes, indicating a lack of cohesion within these classes. High LCOM is problematic because it suggests that methods within the class are not strongly related, which can make the code harder to maintain due to difficulties in understanding and modifying the class without affecting its behavior.

Most classes have low DIT values, which is beneficial as it suggests less complexity and better maintainability. However, the few classes that show higher DIT values could potentially suffer from the negative effects of deep inheritance hierarchies, such as increased difficulty in understanding and modifying the code due to inherited behavior.

CBO shows varying degrees across classes with several higher values indicating that some classes are highly dependent on other classes. High coupling can complicate maintenance tasks because changes in one class may necessitate changes in its dependent classes.

Larger class sizes are associated with higher WMC values, confirming the trend that larger classes tend to be more complex. This suggests that reducing the size of classes may help in managing complexity and thus improve maintainability.

LCOM values do not show a clear trend with class size, indicating that class size alone might not be a significant factor in affecting how cohesive a class is. This might suggest that other factors like the nature of the functionality implemented in the class or the design patterns used could have more influence on cohesion.

The scatter plot suggests an upward trend with larger classes showing higher CBO values. This reinforces the notion that larger classes tend to interact more with other classes, increasing the maintenance burden due to higher coupling.

Based on Correlation Matrix:

The presence of strong positive correlations indicates that there is a direct relationship between the number of lines of code (LOC) in a class and the complexity of the class, as assessed by Weighted Methods per Class (WMC). A correlation coefficient of 0.84 quantifies this relationship. These findings indicate that classes of greater size generally exhibit greater complexity, which can have a detrimental effect on maintainability. Also, the total number of methods (totalMethodsQty) has a strong positive relationship with WMC (0.83), which supports the idea that classes with more methods are likely to be more complicated.

The data shows moderately positive associations between Lack of Cohesion in Methods (LCOM*) and both LOC (0.42) and totalMethodsQty (0.50). This implies that classes with a higher number of lines of code (LOC) and a larger number of methods may have lower cohesion among those methods, which could have a negative impact on maintainability. In addition, the metric known as

Coupling Between Objects (CBO) demonstrates a moderately positive association with two other metrics: Lines of Code (LOC) with a correlation coefficient of 0.62 and totalMethodsQty with a correlation coefficient of 0.42. These findings suggest that larger classes and classes with a greater number of methods are likely to exhibit stronger connections with other classes, resulting in increased complexity and decreased maintainability.

Low correlations indicate that measures such as totalFieldsQty and Depth of Inheritance Tree (DIT) have a limited impact on class size or maintainability in this specific situation. These metrics demonstrate weak correlations with important class size metrics, suggesting that they have a minimal effect on the overall maintainability of classes.

Insights and Implications of Project Java Design Patterns

Based on Bar Chart and Scatter Plot:

The WMC values are generally low, indicating that most classes have manageable complexity. A few peaks do exist, suggesting that specific classes with many methods or complex methods may require more effort to maintain.

LCOM values vary significantly, with many classes showing higher values. This suggests that there is a lack of cohesion in a significant number of classes, which could hinder maintainability by making these classes more difficult to modify and understand.

The DIT values are mostly low, which is beneficial as it implies simpler inheritance structures. Simpler inheritance tends to facilitate easier maintenance and understanding. The occasional higher values may indicate more complex inheritance structures that could complicate maintenance.

CBO values mostly remain moderate, but there are classes with higher values, indicating significant interdependencies. High coupling can make maintenance more challenging as changes in one class might affect multiple other classes.

The scatter plot shows that most classes with larger sizes tend to have higher WMC values. This reaffirms that larger classes generally have more complexity, which may reduce their maintainability.

LCOM values appear relatively scattered regardless of class size, indicating that cohesion within a class is not strongly influenced by its size. This suggests that factors other than size, perhaps the specific responsibilities or design of the class, are more significant in determining cohesion.

CBO values are also moderately scattered, with no clear trend showing an increase with class size. This suggests that while some larger classes exhibit higher coupling, it is not a consistent rule, and coupling is likely influenced by specific class design choices and interactions.

Based on Correlation Matrix:

The data show a significant positive association between the number of lines of code ("loc") in a class and its complexity, as assessed by Weighted Methods per Class ("wmc"). The correlation coefficient between these two variables is 0.87. Similarly, there is a significant positive connection (0.84) between the total number of methods ("totalMethodsQty") and "wmc", suggesting that classes with a greater number of methods are likely to exhibit higher complexity. These findings indicate that classes of a larger size possess inherent complexity, which can have a detrimental effect on their maintainability.

The moderately positive correlations underscore the moderately positive relationship between the variable "totalFieldsQty" and both "loc" and "wmc" (0.48 each). This suggests that classes with a

higher number of fields tend to be larger and more complicated. Additionally, the metric "lcom*" (Lack of Cohesion in Methods) has strong positive relationships with both "totalMethodsQty" (0.50) and "loc" (0.40). This means that classes with more methods and more lines of code may have less cohesion, which can make them harder to maintain. The metric "cbo" (Coupling Between Objects) demonstrates modest positive correlations with "loc" (0.51) and "wmc" (0.40), indicating that larger classes with a greater number of methods are likely to have stronger connections with other classes, which adds complexity to maintenance tasks.

Conversely, only minor connections were found with "dit" (Depth of Inheritance Tree), suggesting that the depth of inheritance may have a limited impact on class size or maintainability in this situation.

Insights and Implications of Project Apache Dubbo

Based on Bar Chart and Scatter Plot:

The WMC values show significant variation across classes, with some classes exhibiting extremely high values. The presence of outliers with very high WMC values indicates that certain classes are highly complex, which can make them difficult to maintain. Generally, a higher WMC suggests greater complexity and potential maintainability challenges.

The LCOM values are distributed broadly, with many classes having high LCOM values, indicating low cohesion. High LCOM values suggest that methods within a class are not well-related, making the class harder to maintain and understand. Cohesion does not appear to strongly correlate with class size, indicating that other factors influence cohesion.

The majority of classes have low DIT values, indicating shallow inheritance hierarchies. Shallow hierarchies are generally easier to maintain due to less complexity and fewer inherited

dependencies. However, there are a few classes with higher DIT values, which could imply more complex inheritance structures that might complicate maintenance.

The CBO values exhibit variation, with some classes showing significantly higher coupling. High CBO values indicate classes that are highly dependent on others, which can increase maintenance difficulty as changes in one class may propagate to others. Managing coupling is essential to maintaining class modularity and reducing maintenance effort.

The scatter plot reveals a positive correlation between class size (LOC) and WMC. Larger classes tend to have higher WMC values, indicating that they are more complex and likely harder to maintain. This suggests that as classes grow in size, their complexity increases, which can negatively impact maintainability.

The scatter plot shows that LCOM values do not have a clear correlation with class size. This indicates that class size alone does not determine cohesion; rather, the design and purpose of the class play a more significant role in its cohesion.

The scatter plot shows some larger classes with high CBO values, but there is not a strong overall trend. This suggests that while larger classes can have higher coupling, this is not a consistent rule. Coupling is more influenced by how a class is designed to interact with other classes.

Based on Correlation Matrix:

The data show a significant positive relationship between the number of lines of code ("loc") and the class's complexity, as measured by the correlation coefficient of 0.80 between LOC and Weighted Methods per Class ("wmc"). This indicates that as the number of lines of code increases, so does the class's complexity. These findings indicate that classes of greater size have a tendency to possess higher levels of complexity, which can have a detrimental effect on the ease of

maintenance. Similarly, the variable "totalMethodsQty" exhibits a robust positive correlation of 0.85 with WMC, suggesting that classes with a greater number of methods generally have higher complexity.

The moderate correlations indicate that there is a positive relationship between "lcom*" (Lack of Cohesion in Methods) and both "totalFieldsQty" (0.66) and WMC (0.47). These findings indicate that classes that have a larger number of fields and methods may have lower cohesiveness, which might negatively impact the ease of maintenance. In addition, the metric known as "cbo" (Coupling Between Objects) shows modest positive relationships with LOC (0.55), totalMethodsQty (0.40), and WMC (0.45). This suggests that classes that are larger in size and have a greater number of methods are more likely to have a stronger connection with other classes, making the process of maintaining them more complex.

In this specific scenario, the data shows weak associations with "dit" (Depth of Inheritance Tree), suggesting that the depth of inheritance may have a limited impact on class size or maintainability. Similarly, the metric "totalFieldsQty" shows poor correlations with other metrics, indicating that the overall number of fields may not have a substantial impact on class size or maintainability in this circumstance.

Insights and Implications of Project Netty

Based on Bar Chart and Scatter Plot:

The WMC values show significant variation, with some classes exhibiting extremely high values. The presence of outliers with very high WMC values (reaching up to 700) indicates that these classes are highly complex, which can negatively impact maintainability. Higher WMC values

suggest that these classes have many methods or methods of high complexity, making them harder to maintain.

The LCOM values are spread broadly across the classes, with many classes showing high LCOM values. High LCOM values indicate low cohesion within the classes, which can complicate maintenance tasks by making the classes harder to understand and modify.

Most classes have low DIT values, indicating shallow inheritance hierarchies. However, there are some classes with higher DIT values (up to 25), which could signify complex inheritance structures. Classes with deep inheritance trees can be harder to maintain due to the increased complexity of understanding and modifying inherited behaviors.

The CBO values vary, with several classes showing significantly higher coupling (up to 70). High CBO values indicate classes that are highly interdependent, which can reduce maintainability as changes in one class might necessitate changes in many others.

The scatter plot reveals a clear positive correlation between class size (LOC) and WMC. Larger classes tend to have higher WMC values, indicating that they are more complex and likely harder to maintain. This trend suggests that as classes grow in size, their complexity increases, making them more challenging to manage.

The scatter plot shows no strong correlation between class size and LCOM. This suggests that class size alone does not determine cohesion; other factors, such as the specific responsibilities or design of the class, play a more significant role in determining cohesion.

While there is some indication that larger classes can have higher CBO values, the trend is not strong. This suggests that coupling is more influenced by how a class is designed to interact with other classes rather than its size alone.

Based on Correlation Matrix:

The data shows a strong positive correlation between the number of lines of code ("loc") and the difficulty of the class, as measured by the correlation coefficient of 0.87 between LOC and Weighted Methods per Class ("wmc"). This means that as the number of lines of code rises, the complexity of the class also increases. These findings indicate that classes of greater size have a tendency to possess higher levels of complexity, which can have a detrimental effect on maintainability. Similarly, the variable "totalMethodsQty" exhibits a significant positive correlation of 0.86 with WMC, suggesting that classes with a greater number of methods tend to possess more complexity.

The data indicates that there are moderately positive relationships between "lcom*" (Lack of Cohesion in Methods) and both "totalFieldsQty" (0.53) and WMC (0.43). This implies that classes with a larger number of fields and methods may have lower cohesiveness, which can impact the ease of maintenance. In addition, the metric "cbo" (Coupling Between Objects) shows modest positive relationships with LOC (0.61), totalMethodsQty (0.47), and WMC (0.51). This suggests that classes that are larger in size and have a greater number of methods are more likely to have a higher level of coupling with other classes, which can make the process of maintaining them more complex.

The "dit" (Depth of Inheritance Tree) metric exhibits moderately positive relationships with metrics such as LOC (0.27), totalMethodsQty (0.23), and WMC (0.24). This implies that as the number of levels in the inheritance hierarchy increases, there might be a modest rise in the size and complexity of the classes, which could potentially affect the ease of maintaining the code to some degree. Nevertheless, the total effect is less significant when compared to other indicators.

Conclusion:

The objective of this study was to examine the influence of class size on the maintainability of software components by employing the Goal-Question-Metric (GQM) approach. The major goal was to investigate the impact of different class sizes on maintainability measures, including complexity, cohesion, inheritance depth, and coupling. An analysis was performed on multiple projects, namely Project Arthas, Nacos, Java Design Patterns, Apache Dubbo, and Netty. Bar charts, scatter plots, and correlation matrices were utilized to extract valuable insights from the data.

Relationship between Complexity (WMC) and Class Size

The analysis repeatedly shown a positive association between class size, which was measured in Lines of Code (LOC), and complexity, which was quantified by Weighted Methods per Class (WMC). Classes with a greater number of students generally had higher Working Memory Capacity (WMC) ratings, which suggests a greater level of complexity. This pattern was apparent in all examined projects. Classes with high Weighted Method Count (WMC) values sometimes have several methods or methods with substantial complexity, resulting in increased difficulty in maintaining them. For instance, the scatter plots demonstrated a positive relationship between class size and WMC (Working Memory Capacity), with correlation values reaching as high as 0.93 in Project Arthas.

The ramifications of these findings are evident: bigger classes are more intricate, which has a detrimental effect on maintainability. Classes with a high level of complexity might pose challenges in terms of comprehension, modification, and testing. Hence, it is imperative to ensure

sustainable complexity levels by effectively regulating class size through the process of refactoring and implementing modular design.

The relationship between cohesion (LCOM) and class size

The Lack of Cohesion in Methods (LCOM) ratings shown significant variation among classes, with a considerable number displaying moderate to high levels of LCOM. This suggests that larger courses frequently have a lack of coherence. Nevertheless, the association between class size and LCOM was less robust compared to WMC. The scatter plots and correlation matrices indicated that there was no consistent positive relationship between LCOM and class size. This implies that factors other than size, such as the distinct obligations and structure of the class, have a greater impact on determining cohesion.

Although the link is smaller, the influence of inadequate cohesion on maintainability is significant. Classes with high LCOM (Lack of Cohesion of Methods) ratings are more challenging to comprehend and alter due to the lack of strong relationships between their methods. This can result in a greater amount of maintenance work and an increased probability of introducing errors when making changes. Enhancing maintainability can be achieved by improving coherence through improved class design and adhering to the single responsibility principle.

Analysis of Inheritance Depth (DIT) and Class Size

The DIT values were predominantly low, suggesting that most projects had shallow inheritance hierarchies. This is advantageous for the sake of maintainability since it suggests reduced intricacy and a decreased number of dependencies. Nevertheless, there were occurrences of classes with elevated DIT values, indicating more complex hierarchies. Classes with high DIT values can be more challenging to maintain and understand since they have complex inherited behaviors.

The association between class size and DIT was found to be minimal, suggesting that there is no meaningful impact of class size on inheritance depth. Nevertheless, it remains crucial to effectively handle the depth of inheritance in order to ensure maintainability. It is advisable to prioritize shallow hierarchies in order to decrease complexity and enhance comprehensibility.

The relationship between Coupling (CBO) and Class Size

The Coupling Between Objects (CBO) values exhibited substantial variation, with several classes exhibiting high coupling. Maintenance can be complicated by the presence of high CBO values, which suggest that there are significant interdependencies between classes. The risk of defects and the effort required for maintenance may be increased as a result of changes in one class necessitating changes in its dependent classes.

The correlation matrices and scatter diagrams indicated a moderate positive correlation between class size and CBO. Larger classes were more likely to engage in interactions with other classes, which resulted in an increase in their coupling. This implies that coupling can be effectively managed by manipulating class size. By minimizing dependencies and designing classes with well-defined interfaces, it is possible to reduce coupling and improve maintainability.

References

1. Aniche, M. (n.d.). mauricioaniche/ck. GitHub. https://github.com/mauricioaniche/ck
2. Alibaba. (n.d.). alibaba/arthas. GitHub. https://github.com/alibaba/arthas
3. Alibaba. (n.d.). alibaba/nacos. GitHub. https://github.com/alibaba/nacos
4. Iluwatar, E. (n.d.). iluwatar/java-design-patterns. GitHub. https://github.com/iluwatar/java-design-patterns
5. Apache. (n.d.). apache/dubbo. GitHub. https://github.com/apache/dubbo
6. Netty. (n.d.). netty/netty. GitHub. https://github.com/netty/netty
7. Pandas Documentation. (n.d.). pandas.DataFrame.corr. pandas. https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.corr.html