SSW567

Impact of Code Duplicates Removal on Quality - Phase I, Phase II

Group 6: Jiayin Huang, Sijie Yu, Rongda Kan, Yulong Yan, XiaoPeng Wu

Introduction

As business applications and software have become more complex and feature-rich, code bases become larger and the resources that are required to run them have grown exponentially over the last two decades. Code quality is important for ensuring reliability and scalability for the change. A great interest has been observed in the literature in the identification of parts of the code that need to be improved. In particular, duplicated code is one of the most pervasive and pungent smells to remove from source code through refactoring. It is common knowledge that code duplicates could have a number of negative impacts on a software system. Such as the potential of running into bugs, large code size, maintenance costs, readability, and usefulness, etc. Existing studies present multiple methodologies to identify and remove code duplicates. However, more quantitative studies are needed to measure the severity of the impact on systems. This would help stakeholders to identify the importance of improving code quality and funnel more research resources to develop better applications that help to remove code duplicates.

We extensively brief through the literature on quality attributes. Duplicated code has been largely studied, and different types of duplications have been identified. Georges divides these duplicated codes into two categories that a complete function is duplicated exactly or only a piece of code is found as part of a number of functions. And he concludes that by eliminating the duplicates, you ensure that the code says everything once and only once, which is a rule of good removal[1]. Some studies have experimented with different refactoring tools to remove code duplication. The refactoring tool that they found more useful during the experimentation is Eclipse and it’s very simple and intuitive to use[2]. Another useful approach and tool developed to suggest code duplicate removal is called Duplicated Code Refactoring Advisor (DCRA) and is composed of four modules[3].

As we continue to investigate existing studies and complete preliminary research, we may identify other perspectives for validating the impact of code duplicates.

It will be essential to use a systematic approach that takes into account both the technical and non-technical facets of software development in order to address the effect of refactoring on the relationship between quality attributes and design metrics.
Under the above circumstances, this paper aims to conduct research on code refactoring with underlying variables. More specifically, our goal is to analyze data generated from code for understanding whether refactoring duplicates could impact code quality and performance by applying methodologies:

1) Data preparation. The sample should be cleaned up and represents the majority of SDLC processes.
2) Identifying and measuring the size of duplicates. We will leverage existing methods, and set up critical quantified metrics that indicate code duplicates in the system.
3) Measuring performance under a controlled experiment. We will compare performance of datasets of both before and after code refactoring. We will continue to investigate the right methods to analyze and measure the severity impact.

Study Design:

Our primary goal is to see if the developer perception of quality improvement (as expected by developers) corresponds to actual quality improvement (as assessed by quality metrics). We specifically address the following research question:

a. *RQ1: Is the developer's perception of duplicate removal aligned with the quantitative assessment of code quality?*
b. *RQ2: What triggers developers to refactor the code for duplicate code removal?*

To answer our research question, we have conducted data analysis and data cleaning form two sets of github commits dataset regarding 7 metrics to analysis if code quality actually improved by comparing the metrics results between before refactoring and after refactoring. In addition we have conducted manual analysis of commit example for 7 metrics to find the interests of refactoring the code for duplilicate code removal.

Before the Data Visualization we Conducted a data cleaning following given rules:

1. Remove Null values, and Outliers from before and after columns.
2. If one cell does have value but other cell does not have value add 0 to the cell that does not contain value.
3. Once finishe above steps, ensure both values from one row have equal data points.

For manuinspection, we intend to navagatived one of the commit from the dataset to explicitly analyze the code that has been factores to investigate the detail why the part of the cod is being refactored and observe the effects it will bring to code base.

Experiments

Following figures are data analysis conducted for flowing metrics (LCOM, CBO, RFC, DIT, NOC, WMC, LOC): code at here[10]

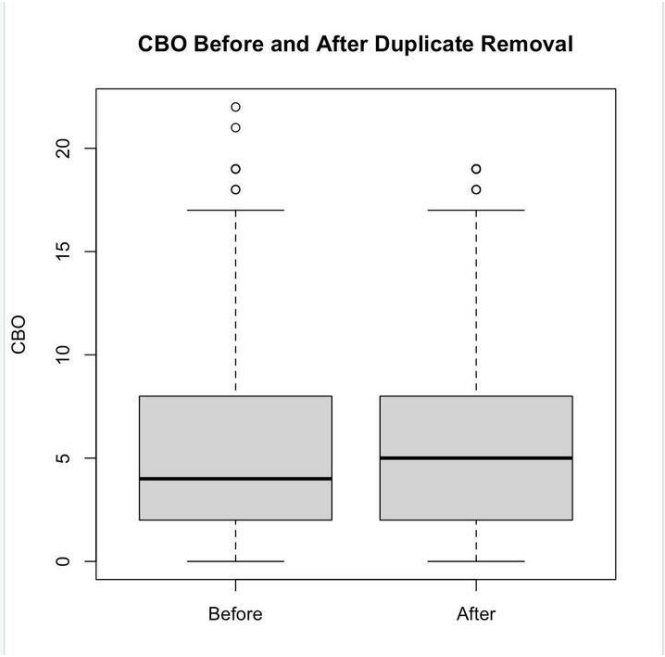

Figure 1a: CBO Set 1

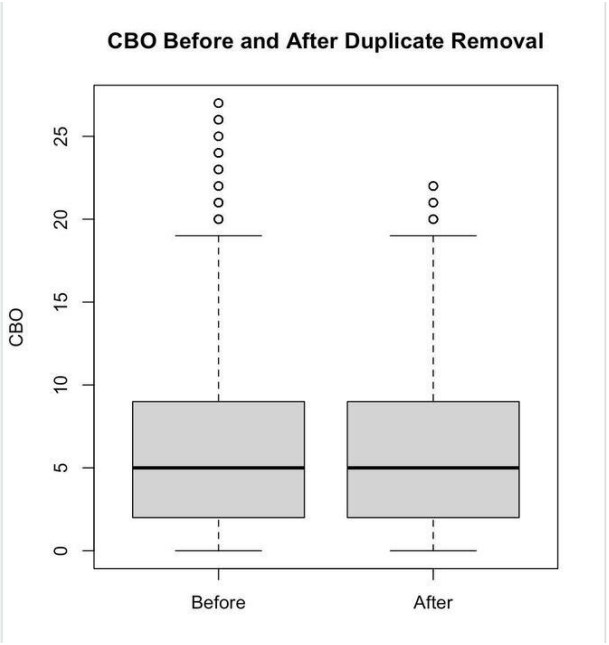

Figure 1b:CBO Set b

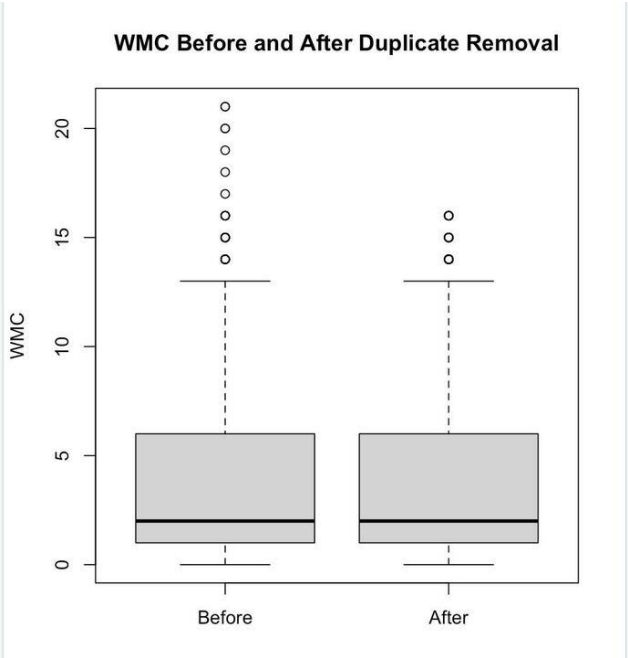

Figure 2a: WMC Set 1

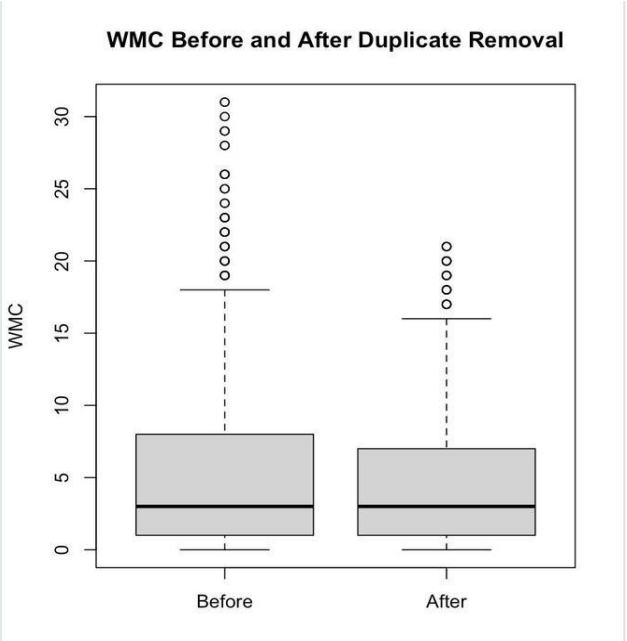

Figure 2b:WMC Set 2

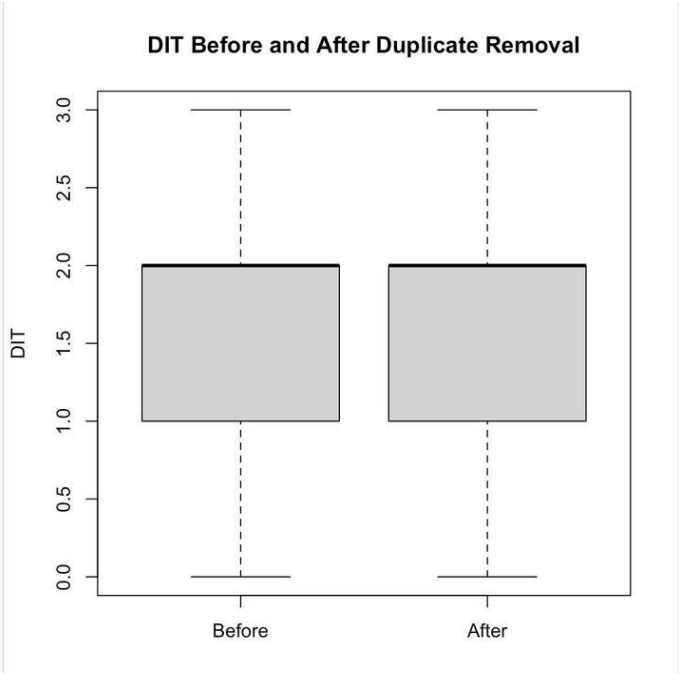

Figure 3a: DIT Set 1

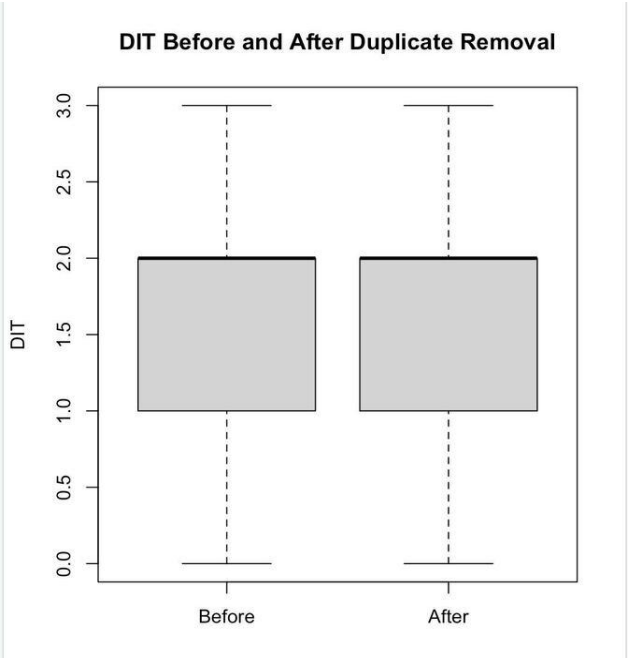

Figure 3b DIT Set 2

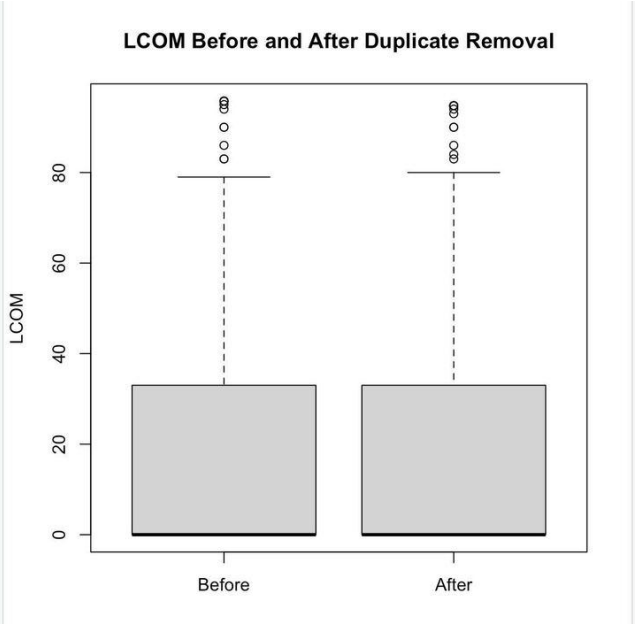

Figure 4b LCOM Set 2

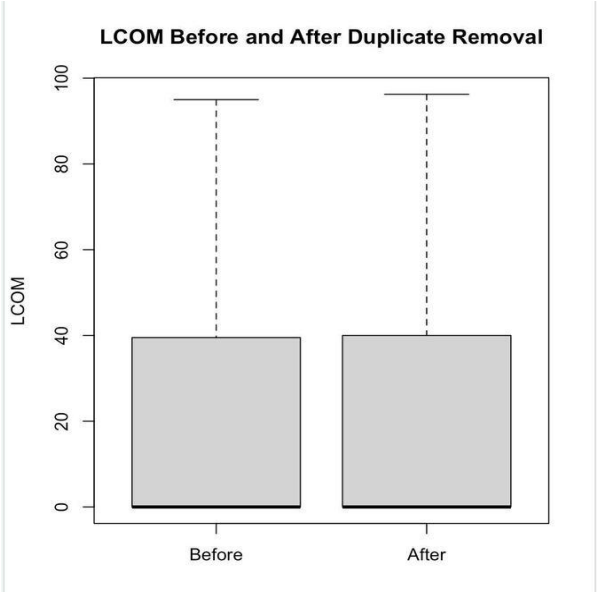

Figure 4a: LCOM Set 1

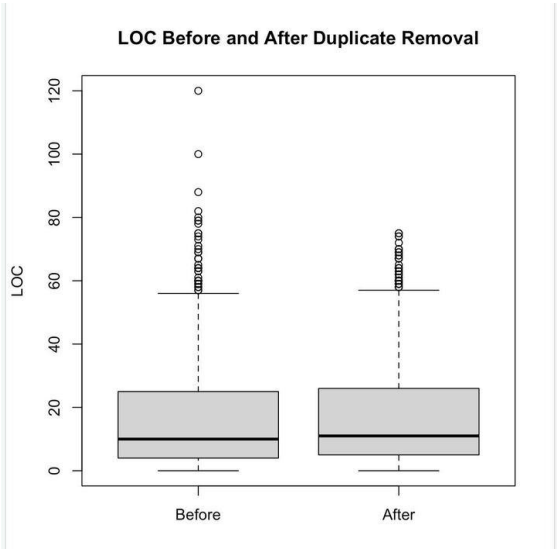

Figure 5a: LOC Set 1

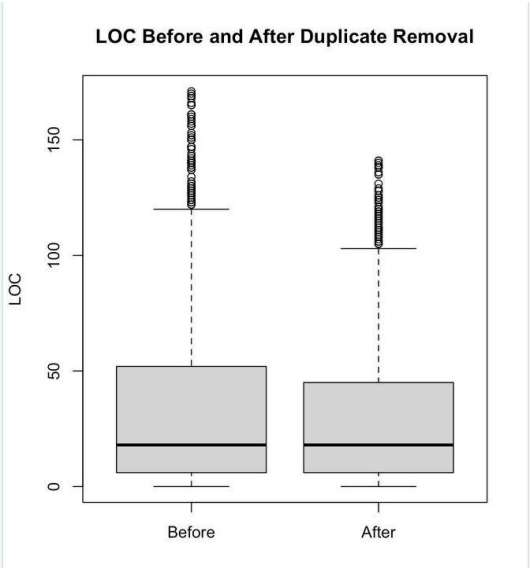

Figure 5b LOC Set 2

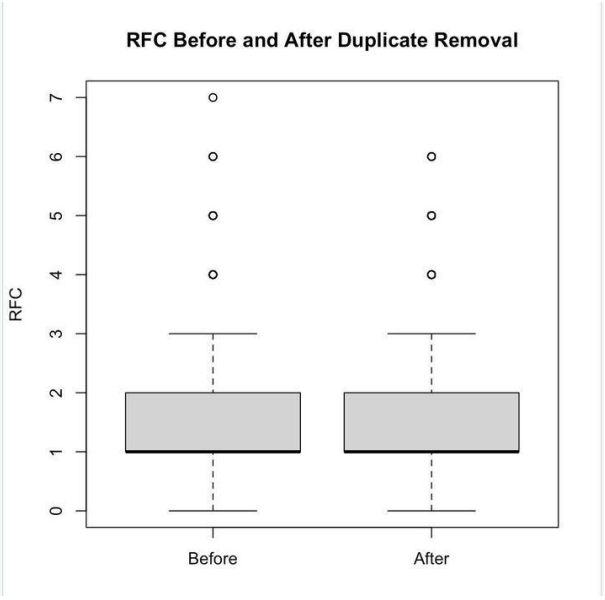

Figure 6a: RFC Set 1

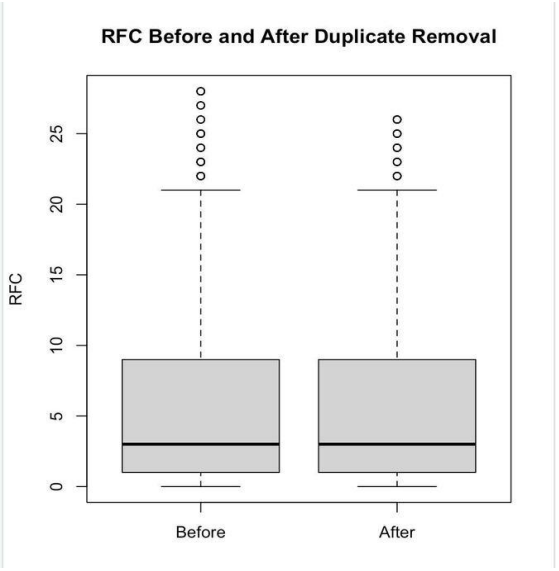

Figure 6b RFC Set 2

Experiment

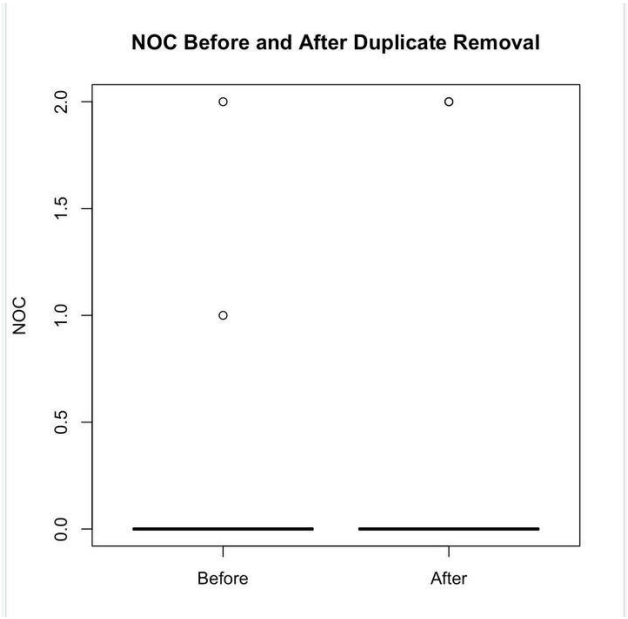

Figure 7a: NOC Set 1

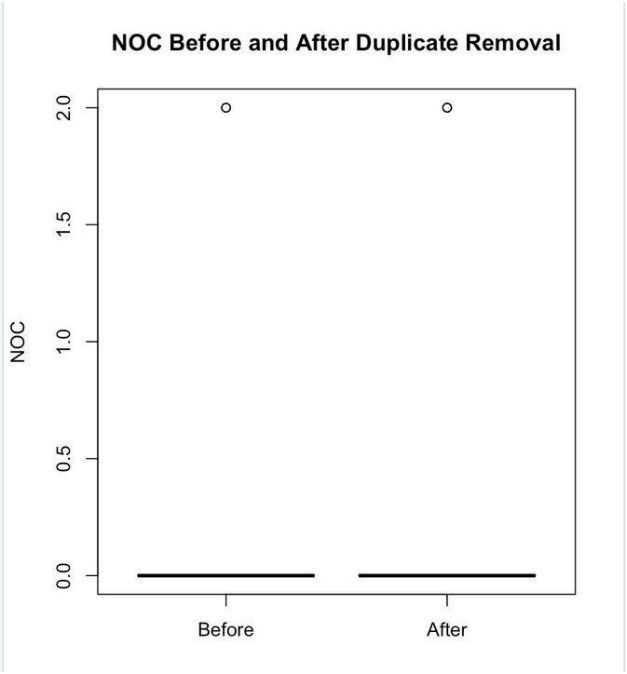

Figure 7b NOC Set 2

1.Coupling:
a. RQ1
As shown in the boxplots in the figure 1a and 1b, we find that all variations in set1 are significant. And the CBO metric experiences a degradation in the median value in the set1, while the CBO has no changes in the median value in the set2.

b. RQ2
Commit[4] that indicate CBO improvement: We could see that the implementation of class JenkinsRssWidget has decoupled an instance of Project that initiated at early phase, using this refractory to reduce usage. So this optimization decreases the complexity significantly

Summary:The CBO metric is used to measure the degree to which one class depends on another class. Combine the CBO metrics boxplots in set1 with that in set2, all the variations are insignificant
[1]https://github.com/MCMicS/jenkins-control-plugin/commit/aa2ee9f6928763113f11a372a09fbb3b6253310c

2.Complexity:

WMC
a.RQ1:
As seen in the boxplots in figure 2a and 2b, we observe that all the variations in the WMC metric are statistically significant. In the analysis of boxplots in set1 and set2, the upper whisker in the set2 boxplots decreases significantly, while in the set1, the upper whisker has no change.
b.RQ2
Commit[5] that indicates WMC improvementThe Method called testCreateLineOfCreditForAClient and testCreateLineOfCreditForAGroup in the class have been merged into one method called testCreateLineOfCreditForAnAccountHolder to simplify the implementation. So this optimization decreases the complexity positively, which is a good trigger for developers to refactor the code.

Summary: The WMC metric is used to measure the complexity of a software program. It's calculated by adding up the complexity of each method in a class, where complexity is defined in terms of the number of decision points in the method. Combine the WMC metrics boxplots in set1 with that in set2, all the variations are significant and the variations in the set2 are much more significant and the upper whisker decreases. From this analysis, we can conclude that the WMC metric decreases as developers intend to improve complexity.

3.Cohesion:
LCOM
a.RQ1: According to the boxplot figure 4a and 4b, the medians of LCOM before and after duplicate removal are both 0, the interquartile ranges are basically same, which means that the discrete cases of LCOM before and after duplicate removal are basically consistent. Also, the maximum numbers of LCOM of before and after are consistent. By this observation, we can tell that the LCOM before and after stays basically unchanged. LCOM is used to measure the cohesion between methods in a class, that is, whether the methods of a class are closely related. If there are few inter-relationships between methods in a class, that is, the methods lack cohesion, then there is probably something wrong with the design of the class and needs to be optimized.
b.RQ2: Commit[6] that indicates the LCOM staying unchanged before and after the duplicate removal. LimitDeparser and OrderByDeParser have been pulled out into separate classes to avoid code duplication from SelectDeParser. This operation does not decrease the LCOM significantly, so there is no reason for developers to do duplicate removal for improving the LCOM.
Summary: The normalized LCOM metric can serve not only as a good surrogate for the original LCOM metric, but also as a proxy for cohesive quality attributes. If the value of the normalized LCOM metric increases, the developer's intent to improve cohesion is achieved.
[3]https://github.com/jsqlparser/jsqlparser/commit/a31333a65141e2753717757b32261761105d384b

4.Inheritance:
NOC and DIT
a.RQ1

As shown in the boxplots in figure 3a and 3b, the DIT metric almost experiences no change compared before and after in both set1 and set2.
But in the boxplots in figure 7a and 7b, the NOC metric experiences an insignificant decrease. Furthermore, all variations are insignificant and the variations in set2 do not change any more.

b.RQ2
Commit[7] that indicates DIT improvement:
https://github.com/apache/httpcomponents-client/commit/1dd6b903748a994f34f11998d6c5ba2dbc8d9810
After the duplicate removal, the developers made AbstractMultipartForm and its superclasses package private. This operation effectively reduces the Depth of Inheritance Tree and improves the code quality of DIT indicators, which means this commit of duplicate removal effectively influences the DIT of code quality.

Summary: The DIT metric is used to measure the depth of inheritance in a class hierarchy. It's calculated by counting the number of ancestor classes between a given class and the root of the class hierarchy. And the NOC metric is used to measure the number of immediate subclasses of a class. It's calculated by counting the number of immediate subclasses of a class. From the above analysis, DIT generally doesn't change and its variations are insignificant. Furthermore, our empirical investigation discards NOC from being an indicator for inheritance.

5.Polymorphism
WMC and RFC
a.RQ1: For WMC in the boxplot figure 2a, The discrete distribution is basically the same, but the outlier of WMC after the duplicate removal decreases significantly. For figure 2b, the medians stay unchanged, but the upper quartile slightly decreases, and also the range 75%-100% decreases. For RFC in the boxplot figure 5a, the boxplot stays basically unchanged. For figure 5b, the data distribution also stays unchanged before and after. The weight of a method in a class refers to the sum of the complexity of each method in a class. If the Weighted Methods per Class is high, it indicates that the method of this class is more complex and needs to be refactored to simplify the method implementation. The RFC of a class refers to the number of methods in a class that are called by other classes. If the value of Response for a Class is high, it indicates that the function of this class is more complex, and splitting or refactoring needs to be considered.

6.Encapsulation
WMC,LCOM
a.RQ1:according to boxplot Figure 4a and Figure 4b, the medians of LCOM in both figures does not have significant differences, which means there are minor changes after refactoring. And for the results that comparing differences between dataSet1 and dataSet2, also not changed much. As for the results of WMC, we could get conclusion from Figure2a and Figure 2b after refactor, there is changes in first 25%quantile of the data, with a minor variation in the median, this might

relate with some of the code that is more sensitive to this kind of modification that affect this metric.
b.RQ2: Commit[8] that indicates WMC improvement
https://github.com/mambu-gmbh/Mambu-APIs-Java/commit/b7320cfd90717a07cbd1f32ad2251fa8e520f686 The Method called testCreateLineOfCreditForAClient and testCreateLineOfCreditForAGroup in the class have been merged into one method called testCreateLineOfCreditForAnAccountHolder to simplify the implementation
Commit[9] that indicates LCOM changes:
https://github.com/tordanik/OSM2World/commit/43596381115d6427f56240ce26ddc6bbd9a0e0d1 under this commitment, the author is doing refactor by avoiding duplicated intersection checks, it's removing the function Map2dTree, swapped with another cleaner function called MapIntersectionGrid, which did major modifications by decoupling the previous functions.

Summary: As developers aim to enhance encapsulation, both WMC and the normalized LCOM tend to decrease, but their changes are not statistically significant. As a result, we were unable to identify any metric with a significant positive variation that corresponds to the developer's goal of improving encapsulation.

7 Design Size:LOC
a.RQ1:The LOC metrics before and after decrease as developers intend to improve design size but the variations are not significant. So LOC metrics have not a significant positive variation which matches the developer's perception of improving design size.
b.RQ2: Commit [] that indicates LOC improvementThe XercesErrHandler function has been deleted and the error function has been refactored so that the LOC has been optimized and the implementation of the function error has been simplified.

[1] Georges G. K. : A Scenario Based Approach for Refactoring Duplicated Code in Object Oriented Systems.
[2] Francesca A. F., Marco M., Domenico P., Marco Z.: On Experimenting Refactoring Tools to Remove Code Smells. XP'15 workshops: Scientific Workshop PRoceedings of the XP2015.
[3] Francesca A. F., Marco Z., Francesco Z.: A Duplicated Code Refactoring Advisor. Agile Processes in Software Engineering and Extreme Programming pp 3-14.
[4]Source Retrived from
https://github.com/MCMicS/jenkins-control-plugin/commit/aa2ee9f6928763113f11a372a09fbb3b6253310c
[5] Source Retrived from
:https://github.com/mambu-gmbh/Mambu-APIs-Java/commit/b7320cfd90717a07cbd1f32ad2251fa8e520f686
[6] Source Retrived from :
https://github.com/jsqlparser/jsqlparser/commit/a31333a65141e2753717757b32261761105d384b

[7] Source Retrived from :
https://github.com/apache/httpcomponents-client/commit/1dd6b903748a994f34f11998d6c5ba2dbc8d9810
[8] Source Retrived from
:https://github.com/mambu-gmbh/Mambu-APIs-Java/commit/b7320cfd90717a07cbd1f32ad2251fa8e520f686
[9] Source Retrived from :
https://github.com/tordanik/OSM2World/commit/43596381115d6427f56240ce26ddc6bbd9a0e0d1
[10] https://github.com/SE4AIResearch/SSW567-Spring2023-Group6