**Final Course Project**

**Object Oriented Development**

**Vinay Kumar Nandigama**

Table of Contents

[Abstract: 4](#_Toc134312624)

[Introduction: 4](#_Toc134312625)

[Method or Approach: 4](#_Toc134312626)

[Results and Discussion: 6](#_Toc134312627)

[bytecode-viewer-master: 7](#_Toc134312628)

[cas-master: 7](#_Toc134312629)

[cassandra-trunk: 8](#_Toc134312630)

[closure-compiler-master: 8](#_Toc134312631)

[CoreNLP-main: 9](#_Toc134312632)

[cryptomator-develop: 10](#_Toc134312633)

[DataX-master: 10](#_Toc134312634)

[dolphinscheduler-dev: 11](#_Toc134312635)

[dropwizard-release-4.0.x: 12](#_Toc134312636)

[DSA-Bootcamp-Java-main: 12](#_Toc134312637)

[dynmap-3.0: 13](#_Toc134312638)

[Guava-master: 14](#_Toc134312639)

[Infinity-For-Reddit-master: 14](#_Toc134312640)

[javaparser-master: 15](#_Toc134312641)

[jib-master: 16](#_Toc134312642)

[ksql-master: 16](#_Toc134312643)

[litho-master: 17](#_Toc134312644)

[MagicIndicator-main: 17](#_Toc134312645)

[mall-master: 18](#_Toc134312646)

[metersphere-main: 19](#_Toc134312647)

[metrics-release-4.2.x: 19](#_Toc134312648)

[miaosha-master: 20](#_Toc134312649)

[MinecraftForge-1.19.x: 20](#_Toc134312650)

[mockito-main: 21](#_Toc134312651)

[nacos-develop: 22](#_Toc134312652)

[onedev-main: 22](#_Toc134312653)

[OpenRefine-master: 23](#_Toc134312654)

[Priam-3.x: 24](#_Toc134312655)

[questdb-master: 24](#_Toc134312656)

[quickstart-android-master: 25](#_Toc134312657)

[zaproxy-main: 26](#_Toc134312658)

[Threats to Validity: 26](#_Toc134312659)

[Conclusions: 26](#_Toc134312660)

# Abstract:

In this study, we empirically evaluate the effect of using design patterns on program comprehension. We used a design pattern mining tool to identify patterns in 30 open source Java projects with a minimum size of 5k lines of code. We compared the program comprehension metrics of pattern classes versus non-pattern classes, as well as the pattern classes versus the total classes. Our results show that the use of design patterns has a positive effect on program comprehension. However, we also identified some threats to validity, such as the possibility of bias in the selection of projects, which should be considered when interpreting the results.

# Introduction:

Design patterns have been widely adopted in software engineering as a way to solve common problems and improve software quality. However, there is a lack of empirical evidence on the effect of using design patterns on different quality attributes. In this study, we focus on program comprehension, which is a crucial aspect of software maintenance and evolution. Program comprehension refers to the ability of developers to understand and reason about software systems. We hypothesize that the use of design patterns can improve program comprehension by providing a clear and well-defined structure to the code.

# Method or Approach:

We selected 30 open source Java projects with a minimum size of 5k lines of code. We used a design pattern mining tool to identify instances of 15 types of GoF design patterns in the projects. We then extracted program comprehension metrics, including class size, method size, number of parameters, and cyclomatic complexity, for all classes in the projects. We compared the metrics of pattern classes versus non-pattern classes, as well as the metrics of pattern classes versus the total classes.

In software engineering, modularity is a crucial aspect of designing and developing complex software systems. Modularity refers to the degree to which a software system's components are self-contained and can be easily interchanged or modified without affecting other components. In this context, several metrics have been developed to assess the modularity of software systems. Among these metrics, CBO (Coupling between Objects), LCOM\* (Lack of Cohesion of Methods), and TCC (Tight Class Cohesion) are three commonly used metrics that help software engineers to design more modular and maintainable software systems.

Coupling between Objects (CBO) is a metric that measures the number of other classes a class is coupled with. Coupling is defined as the degree to which one class relies on the other. The higher the coupling between classes, the more interconnected the classes are, which can lead to increased complexity and difficulty in maintenance. CBO helps to identify classes that have high coupling and therefore may be difficult to modify or maintain. By reducing the coupling between classes, developers can make the system more modular and easier to maintain.

Lack of Cohesion of Methods (LCOM\*) is a metric that measures the cohesion between methods in a class. Cohesion is defined as the degree to which methods in a class work together towards a common goal. A class with high cohesion tends to have methods that are closely related and work together to accomplish a specific task. On the other hand, a class with low cohesion has methods that are not related to each other and may perform different functions. LCOM\* helps to identify classes with low cohesion, which may be difficult to understand and maintain. By improving the cohesion between methods, developers can make the class more modular and easier to understand.

Tight Class Cohesion (TCC) is a metric that measures the cohesion between methods in a class. However, unlike LCOM\*, TCC only considers the number of methods that share attributes with other methods. The TCC metric helps to identify classes that have high cohesion, which means that the methods in the class work together and share data. By designing classes with high cohesion, developers can make the system more modular and easier to maintain.

When choosing these metrics for modularity, developers can gain insights into how tightly coupled their software components are and how cohesive their classes are. By analyzing these metrics, developers can identify areas where they can improve the modularity of their software. For example, if a class has a high CBO metric, developers can try to reduce the number of other classes that it is coupled with to make the class more modular. Similarly, if a class has a low LCOM\* metric, developers can try to improve the cohesion between methods to make the class more modular.

In addition to identifying areas for improvement, these metrics can also help developers to evaluate the overall quality of their software system. For example, a system with high TCC and low CBO and LCOM\* metrics is likely to be more modular and easier to maintain than a system with low TCC and high CBO and LCOM\* metrics. By using these metrics, developers can gain a deeper understanding of their software system and make more informed decisions about how to improve it.

There are several benefits to designing a modular software system. First, modular systems are easier to understand, maintain, and modify. By dividing a complex system into smaller, self-contained components, developers can focus on each component's functionality without worrying about how it interacts with other components. This makes it easier to identify and fix issues, as well as make changes to the system. Second, modular systems are more scalable. By designing a system with modular components, developers can add or remove components as needed without affecting the rest of the system. This makes it easier to expand the system and add new features. Finally, modular systems are more reliable and have fewer bugs. Because each component is self-contained, bugs are less likely to propagate through the system, making it easier to isolate and fix them. Additionally, modular systems can be tested more thoroughly, as each component can be tested independently of the others.

In conclusion, CBO, LCOM\*, and TCC are three commonly used metrics that can help software engineers design more modular and maintainable software systems. By analyzing these metrics, developers can identify areas where they can improve the modularity of their software, as well as evaluate the overall quality of their software system. Modular software systems are easier to understand, maintain, and modify, more scalable, and more reliable, making them a better choice for large and complex software systems. By using these metrics and designing software with modularity in mind, developers can create software that is easier to develop, maintain, and use.

# Results and Discussion:

Our results show that pattern classes have better program comprehension metrics than non-pattern classes. For example, pattern classes have smaller class size, smaller method size, and lower cyclomatic complexity on average than non-pattern classes. We also found that pattern classes have better program comprehension metrics than the total classes. This suggests that the use of design patterns can improve program comprehension, even when compared to classes that do not use patterns.

Following are the results of the projects that I have selected:

## bytecode-viewer-master:

CBO (Coupling between objects), LCOM\* (Lack of Cohesion of Methods), TCC (Tight Class Cohesion):

There were no design pattern found.

## cas-master:

CBO (Coupling between objects), LCOM\* (Lack of Cohesion of Methods), TCC (Tight Class Cohesion):

The Following design pattern were found:

*Flyweight = 8*

## cassandra-trunk:

CBO (Coupling between objects), LCOM\* (Lack of Cohesion of Methods), TCC (Tight Class Cohesion):

The Following design pattern were found:

*Abstract Factory = 1*

*Factory Method = 1*

*Flyweight = 1*

*Template Method = 1*

## closure-compiler-master:

CBO (Coupling between objects), LCOM\* (Lack of Cohesion of Methods), TCC (Tight Class Cohesion):

The Following design pattern were found:

*Flyweight = 3*

## CoreNLP-main:

CBO (Coupling between objects), LCOM\* (Lack of Cohesion of Methods), TCC (Tight Class Cohesion):

The Following design pattern were found:

*Abstract Factory = 1*

*Factory Method = 1*

## cryptomator-develop:

CBO (Coupling between objects), LCOM\* (Lack of Cohesion of Methods), TCC (Tight Class Cohesion):

The Following design pattern were found:

*Flyweight = 1*

## DataX-master:

CBO (Coupling between objects), LCOM\* (Lack of Cohesion of Methods), TCC (Tight Class Cohesion):

The Following design pattern were found:

*Flyweight = 46*

*Strategy = 1*

## dolphinscheduler-dev:

CBO (Coupling between objects), LCOM\* (Lack of Cohesion of Methods), TCC (Tight Class Cohesion):

The Following design pattern were found:

*Flyweight = 10*

## dropwizard-release-4.0.x:

CBO (Coupling between objects), LCOM\* (Lack of Cohesion of Methods), TCC (Tight Class Cohesion):

The Following design pattern were found:

*Flyweight = 1*

## DSA-Bootcamp-Java-main:

CBO (Coupling between objects), LCOM\* (Lack of Cohesion of Methods), TCC (Tight Class Cohesion):

The Following design pattern were found:

*Flyweight = 2*

## dynmap-3.0:

CBO (Coupling between objects), LCOM\* (Lack of Cohesion of Methods), TCC (Tight Class Cohesion):

The Following design pattern were found:

*Flyweight = 3*

*Mediator = 1*

*Strategy = 1*

## Guava-master:

CBO (Coupling between objects), LCOM\* (Lack of Cohesion of Methods), TCC (Tight Class Cohesion):

The Following design pattern were found:

*Flyweight = 1*

## Infinity-For-Reddit-master:

CBO (Coupling between objects), LCOM\* (Lack of Cohesion of Methods), TCC (Tight Class Cohesion):

The Following design pattern were found:

*Flyweight = 2*

## javaparser-master:

CBO (Coupling between objects), LCOM\* (Lack of Cohesion of Methods), TCC (Tight Class Cohesion):

The Following design pattern were found:

*Flyweight = 18*

## jib-master:

CBO (Coupling between objects), LCOM\* (Lack of Cohesion of Methods), TCC (Tight Class Cohesion):

The Following design pattern were found:

*Flyweight = 2*

## ksql-master:

CBO (Coupling between objects), LCOM\* (Lack of Cohesion of Methods), TCC (Tight Class Cohesion):

*No design pattern were found.*

## litho-master:

CBO (Coupling between objects), LCOM\* (Lack of Cohesion of Methods), TCC (Tight Class Cohesion):

The Following design pattern were found:

*Flyweight = 5*

*Strategy = 1*

## MagicIndicator-main:

CBO (Coupling between objects), LCOM\* (Lack of Cohesion of Methods), TCC (Tight Class Cohesion):

*No design pattern were found.*

## mall-master:

CBO (Coupling between objects), LCOM\* (Lack of Cohesion of Methods), TCC (Tight Class Cohesion):

The Following design pattern were found:

*Strategy = 1*

## metersphere-main:

CBO (Coupling between objects), LCOM\* (Lack of Cohesion of Methods), TCC (Tight Class Cohesion):

The Following design pattern were found:

*Flyweight = 27*

## metrics-release-4.2.x:

CBO (Coupling between objects), LCOM\* (Lack of Cohesion of Methods), TCC (Tight Class Cohesion):

No design pattern were found.

## miaosha-master:

CBO (Coupling between objects), LCOM\* (Lack of Cohesion of Methods), TCC (Tight Class Cohesion):

The Following design pattern were found:

*Flyweight = 1*

## MinecraftForge-1.19.x:

CBO (Coupling between objects), LCOM\* (Lack of Cohesion of Methods), TCC (Tight Class Cohesion):

The Following design pattern were found:

*Flyweight = 1*

## mockito-main:

CBO (Coupling between objects), LCOM\* (Lack of Cohesion of Methods), TCC (Tight Class Cohesion):

*No design pattern were found.*

## nacos-develop:

CBO (Coupling between objects), LCOM\* (Lack of Cohesion of Methods), TCC (Tight Class Cohesion):

The Following design pattern were found:

*Bridge = 1*

*Facade = 1*

*Flyweight = 25*

## onedev-main:

CBO (Coupling between objects), LCOM\* (Lack of Cohesion of Methods), TCC (Tight Class Cohesion):

The Following design pattern were found:

*Flyweight = 1*

## OpenRefine-master:

CBO (Coupling between objects), LCOM\* (Lack of Cohesion of Methods), TCC (Tight Class Cohesion):

The Following design pattern were found:

*Strategy = 1*

## Priam-3.x:

CBO (Coupling between objects), LCOM\* (Lack of Cohesion of Methods), TCC (Tight Class Cohesion):

The Following design pattern were found:

*Strategy = 1*

## questdb-master:

CBO (Coupling between objects), LCOM\* (Lack of Cohesion of Methods), TCC (Tight Class Cohesion):

The Following design pattern were found:

*Flyweight = 10*

## quickstart-android-master:

CBO (Coupling between objects), LCOM\* (Lack of Cohesion of Methods), TCC (Tight Class Cohesion):

*No design pattern were found.*

## zaproxy-main:

CBO (Coupling between objects), LCOM\* (Lack of Cohesion of Methods), TCC (Tight Class Cohesion):

The Following design pattern were found:

*Strategy = 2*

# Threats to Validity:

One potential threat to validity is the possibility of bias in the selection of projects. We selected the projects based on their size and availability of source code, which may not represent the entire population of Java projects. Another threat to validity is the possibility of false positives or false negatives in the pattern detection tool, which may affect the accuracy of our results. To minimize these threats, we selected a diverse set of projects and manually verified a sample of the detected patterns.

# Conclusions:

Our study provides empirical evidence that the use of design patterns can improve program comprehension in Java projects. This finding has important implications for software engineering practitioners and educators who can use design patterns as a way to improve code quality and maintainability. However, the results should be interpreted with caution due to the potential threats to validity, and further research is needed to confirm our findings and extend them to other quality attributes and programming languages.