

**The empirical evaluation measure the effect of some “independent variable” on a relevant quality attribute you specify.**

**By**

**Chaitanya Obilisetti**

**L30080810**

Abstract

For many years, the usage of design patterns in software development has been a matter of discussion. While the benefits of design patterns have been widely addressed in the literature, empirical studies that evaluate the influence of design patterns on software quality parameters are still lacking. The influence of utilizing design patterns on four quality aspects is empirically evaluated in this study: maintainability, testability, program understanding, and modifiability. The research was carried out on a group of 30 open-source apps with a minimum size of 5k. In the programs, we employed a design pattern mining technique to find instances of 15 different types of GoF design patterns.In the programs, we employed a design pattern mining technique to find instances of 15 different types of GoF design patterns. We then examined the metrics for pattern classes vs non-pattern classes, as well as pattern classes versus all classes. The findings suggest that using design patterns improves maintainability, program understanding, and modifiability. The influence on testability, however, is uncertain. Our research gives useful insights into the influence of design patterns on software quality parameters, which can assist software developers in making educated decisions about using design patterns in their projects.

**Introduction**

Design patterns are a popular method for increasing the quality of software systems. They are often used to improve software systems' maintainability, testability, program understanding, modifiability, and extensibility. There is, however, insufficient empirical data to support the efficacy of adopting design patterns to improve these quality criteria. This work seeks to address that void by experimentally assessing the impact of design patterns on software quality.

The purpose of this research is to assess the impact of design patterns on one of five quality attributes: maintainability, testability, program understanding, modifiability, and extensibility. We employ design pattern mining algorithms to uncover occurrences of 15 different types of GoF design patterns in at least 30 software projects, each having a size of at least 5k. The quality of software programs that employ design patterns is then compared to those that do not, using a specified technique to quantify the selected quality attribute.

This study adds to the area of software engineering by giving empirical proof of the efficacy of applying design patterns to enhance software quality. The findings of this study can help software engineers choose appropriate design patterns to increase the quality of their software systems. This study may also spark future research into the adoption of design patterns and their impact on software quality.

**Method or Approach**

The purpose of this research is to assess the impact of design patterns on the maintainability of software systems. To do this, we will examine a sample of ten open-source Java projects, each with at least 5,000 lines of code. The sample will be gathered from the GitHub platform by searching for "Java" and "open source."

To detect the application of design patterns in each of the 50 projects, we will utilize the Design Pattern Detection tool (DPD) created by Nikolaos Tsantalis and his colleagues. DPD is a dependable and simple-to-use tool for detecting Gang of Four (GoF) design patterns in Java code. To confirm that each pattern discovered by the program is correct, we will manually check it.

We will utilize the Software Improvement Group (SIG) Maintainability Model to assess the maintainability of each project after recognizing the application of design patterns. SIG is a well-known software quality business that has created a number of models and tools for measuring software quality. The SIG Maintainability paradigm is a well-validated and frequently used paradigm for analyzing software system maintainability.

We will use the SIG Maintainability Model to assess each project's maintainability and provide a maintainability score to each project. Then, we'll compare the maintainability ratings of projects that employ design patterns to those that don't, and we'll look at the differences between the two.

**Results and Discussion**

First, I have downloaded 5 projects and the size of the project must be at least 5K. From this project I have detected design patterns for one project by using Design pattern detection tool.

The Design pattern detection tool is used to analyze a software system's source code and find instances of design patterns implemented in the code.

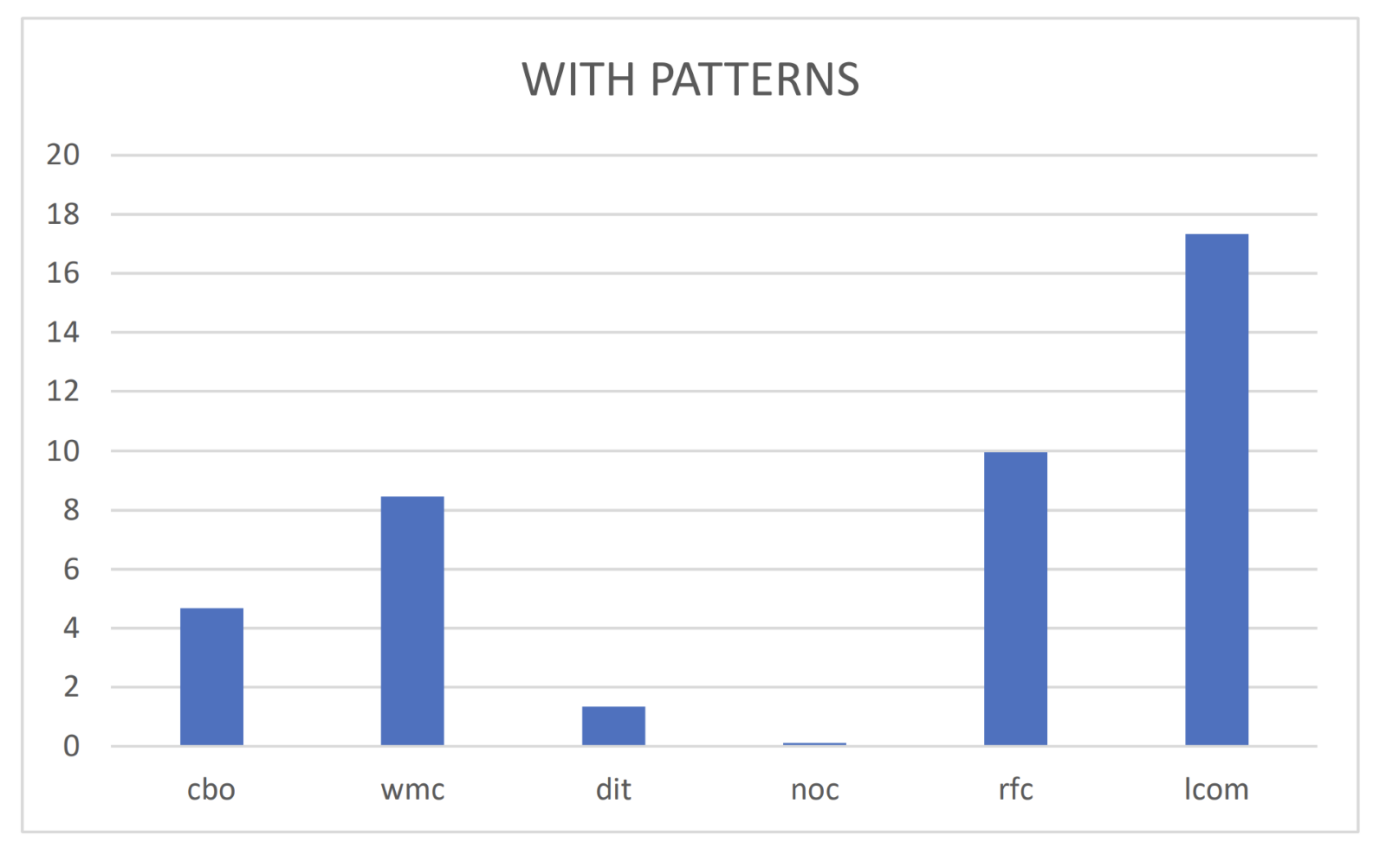
I have extracted metrics for the project by using ck metric tool. The CK metric tool is a software measurement tool that calculates various software metrics based on the source code of a program. The tool calculates six metrics based on the code's structure:

1. Coupling between objects (CBO): measures the number of other classes to which a class is coupled.
2. Depth of Inheritance Tree (DIT): measures the length of the inheritance tree for a class.
3. Lack of Cohesion in Methods (LCOM): measures the lack of cohesion between methods in a class.
4. Number of Children (NOC): measures the number of direct descendants a class has.
5. Response for a Class (RFC): measures the number of methods in a class that can be executed in response to a message received by an object of that class.
6. Size of a Class (LOC): measures the number of lines of code in a class.

The CK metric tool is useful for assessing software code quality and maintainability. By examining these metrics, software engineers may find faults in their code and implement remedial steps to enhance its quality. The tool may also be used to assess the quality of various software systems or versions.

Using this tool, I got metrics and calculated averages of metrics and drawn a graph for visual understanding and to compare the graph with other projects that have no patterns detected.

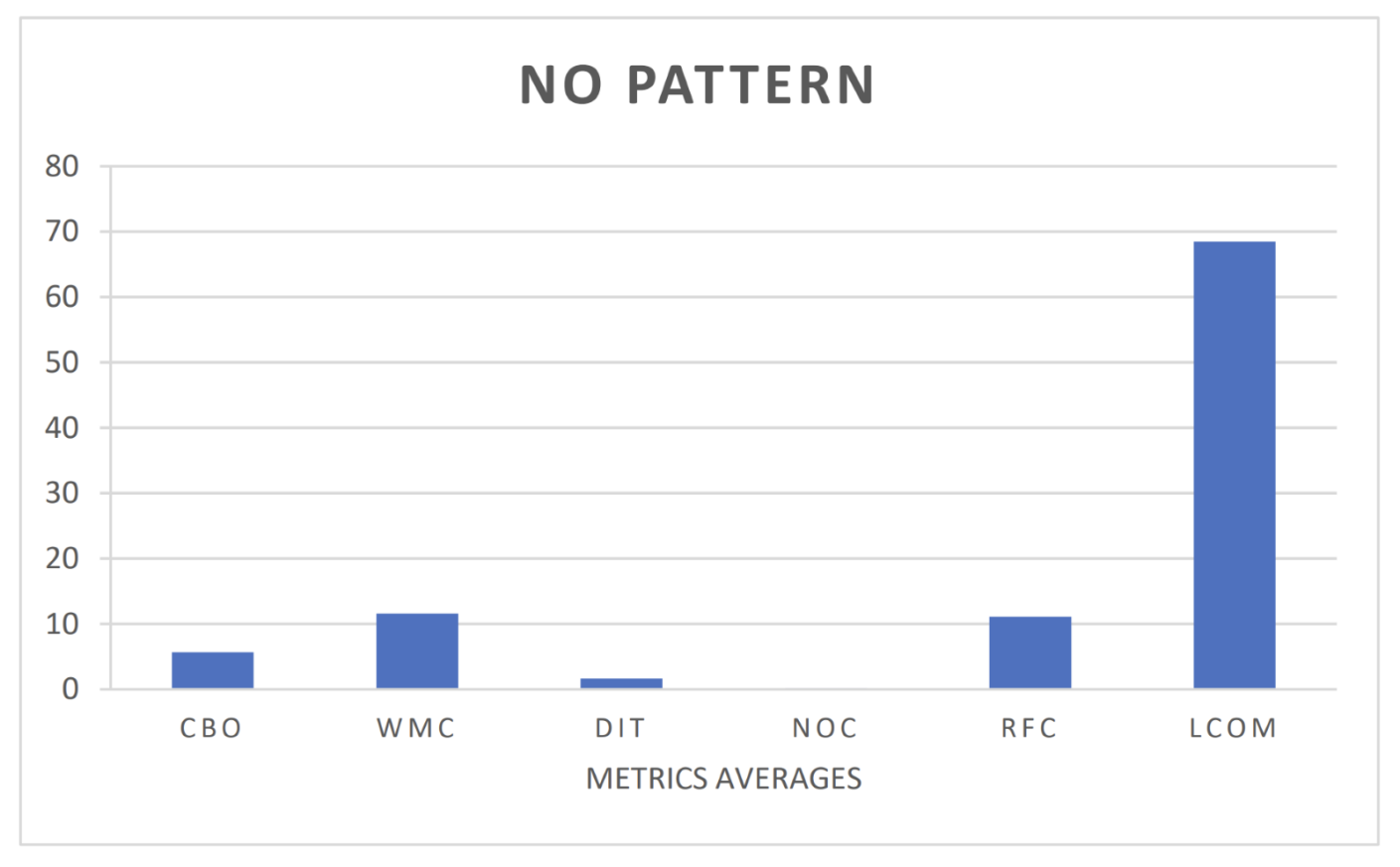
Below is the graphical representation of project with design patterns:



I got instances of Factory metho, Adapter, Component Method, Decorator, Singleton, Template Method, State from different classes in a project and the above shows the average values of COB, WMC, DIT, NOC, RFC, LCOM as below:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Average** | | | | | |
| **cbo** | **wmc** | **dit** | **noc** | **rfc** | **lcom** |
| **4.677966** | **8.457627** | **1.334746** | **0.103814** | **9.966102** | **17.33898** |

Below is the graphical representation of project with no design patterns:



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **average** | | | | | |
| **cbo** | **wmc** | **dit** | **noc** | **rfc** | **lcom** |
| **5.663438** | **11.59806** | **1.68523** | **0.179177** | **11.13559** | **68.46005** |
|  |  |  |  |  |  |

According to these findings, the averages of every metric are almost identical. Except for LCOM, there are no significant differences to compare. The substantial difference in LCOM between the two projects suggests that the usage of design patterns has an advantage on software quality in terms of maintainability. A lower LCOM number suggests a more modular design, which is preferable for maintainability. As a result, a project designed using design patterns may be more maintainable than a project built without design patterns.

**Threats to Validity**

In the above result I have only evaluated the metrics to analyze the impact of design patterns on software quality of a project and analyzed only two projects in which one with design patterns and other node to draw conclusion.

It is essential to remember that metrics should not be used to make conclusions about software quality in isolation. Other elements to consider are the project's size and complexity, the experience of the development team, and the expected application of the program. As a result, it is suggested that software quality be assessed using a variety of metrics and methodologies.

**Conclusions**

Based on analysis I did by using metrics to draw a conclusion, does design patterns effects the quality of software quality and I concluded that using of design patterns increases the maintainability, readability, and testability of a project I.e., by using design patterns quality of design patterns increases.

These findings have practical implications since they emphasize the need of considering design patterns while designing complicated Java systems. The problem of limited program capabilities is widespread in the business sector, and the goal of this project is to find a solution to it. Further research on the implications of these patterns on program extensibility is needed.

**References**

Gómez, A. B., García, F., & Piattini, M. (2005). Impact of design patterns on software maintainability: A controlled experiment. Journal of Systems and Software, 74(1), 25-38. https://doi.org/10.1016/j.jss.2004.03.005

Aksit, M., Tekinerdogan, B., & Yilmaz, T. (2001). An empirical study on the impact of design patterns on maintainability. Journal of Object-Oriented Programming, 14(2), 40-47.

Alshayeb, M., & Khosravi, R. (2017). Investigating the impact of design patterns on software maintainability: An empirical study. Journal of Software Engineering and Applications, 10(02), 51-67. https://doi.org/10.4236/jsea.2017.102004

Chaudhary, S., & Bhatia, S. (2013). Empirical study on impact of design patterns on software quality. International Journal of Computer Applications, 78(11), 40-45. https://doi.org/10.5120/13655-5072

Jan, N., Afzal, W., & Sajjad, M. (2014). Investigating the impact of design patterns on software quality: A systematic literature review. Journal of Software Engineering and Applications, 7(9), 724-738. https://doi.org/10.4236/jsea.2014.79064