Impact of Design Patterns on the Software Testability– An Empirical Analysis

Mallikarjun Ghanapuram

mallikarjunghanapu@lewisu.edu

Kumbham Saiteja

[saitejakumbham@lewisu.edu](mailto:saitejakumbham@lewisu.edu)

Amanaganti Karthik

karthikamanaganti@lewisu.edu

Arun Bhattiprolu

arunbhattiprolu@lewisu.edu

***Abstract*—This study investigates the impact of design patterns on software testability, specifically examining how the use of different patterns affects the ease or difficulty of testing software components. The software testability may be impacted as the number of design patterns or project size increases, leading to an increase in the number of classes and tighter coupling between them.**

**To conduct this empirical study, a dataset of software programs was obtained from GitHub based on specific matching criteria. The CK tool was then used to mine this dataset, collecting various metrics from the classes, methods, and variables in both pattern and non-pattern classes. These metrics were compared between the two groups to gain insight into the testability of software components.**

**The results of this study are presented, providing support for the conclusions drawn. This study contributes to a better understanding of the relationship between design patterns and software testability, providing insights for software developers and researchers to improve software quality.**

**Empirical study will be used by searching through the different programs in the github as per the matching criteria as dataset. This dataset will be mined on both Pattern and Non-Pattern Classes using the CK tool, which collects the different metrics from the Classes, methods, and variables.**

**Keywords— Design patterns, Testability**

I**. INTRODUCTION**

|  |
| --- |

Software testability is a crucial quality attribute that determines the flexibility and ease of testing a software product. It is essential to ensure that the software can be thoroughly tested to identify and resolve any defects or issues that may arise during its development lifecycle. In recent years, the use of design patterns has become increasingly popular in software development. Design patterns are reusable solutions to commonly occurring problems in software design that enable developers to create more maintainable, flexible, and reusable software systems. However, the impact of design patterns on software testability remains an area of ongoing research.

This study aims to understand the impact of design patterns on software testability by analyzing the testability of software components that use different design patterns. Specifically, this study focuses on two widely used design patterns, namely the Singleton pattern and the Adapter pattern. These patterns were chosen for analysis because they are commonly used in software development, and they exhibit different levels of complexity that can potentially affect testability.

To achieve the goal of this study, we followed the Goal-Question-Metric (GQM) approach.

Goal: To evaluate the effectiveness of using design patterns on software testability.

Questions:

1) What is the impact of the Singleton pattern on software testability?

2) What is the impact of the Adapter pattern on software testability?

Metrics: To answer these questions, we selected two metrics, namely the number of children and the depth of inheritance, to measure the impact of design patterns on software testability. The selection of these metrics is based on their ability to provide a quantitative measure of the complexity of the software components and their impact on testability.

# **METHOD OR APPROACH**

Subject programs are the data set which is used to do the analysis or measure the effect using the data provided.

*A. Data Collection and Mining:*

The subject programs used in this study were gathered from the Github platform, which hosts a vast collection of software projects. The projects selected for this study were chosen based on the following criteria: a minimum size of 10MB and a minimum age of 5 years. These criteria were chosen to ensure that the selected projects were mature enough and had undergone several revisions, making them suitable for analysis.

In total, 30 projects were collected from Github based on the above criteria. To identify the design patterns present in the collected projects, the Design Patterns Detection Using Similarity Scoring (DPSS) method was employed [1]. The DPSS algorithm uses directed graphs represented by quadratic matrices and a similarity score approach to characterize all parts of design patterns. By taking the system and the graph of the design pattern as input, it computes the similarity scores between the graph's vertices. The main advantage of this approach is its ability to detect not only the original design patterns as found in the literature but also their modified versions.

After the design patterns were identified in the selected projects, the CK tool was used to collect metrics data from the code. The CK tool, named after its ability to calculate the classLevelMetrics, is a software tool that can calculate various metrics related to software quality . The tool can measure metrics such as Coupling Between Objects (CBO), Fan-In, Fan-Out, Depth of Inheritance Tree (DIT), and Number of Children (NOC), among others. The CK tool is simple to use, and the metrics are provided in CSV format.

To understand the impact of design patterns on software testability, metrics were collected from both the pattern classes and all classes and then compared with each other. By comparing the metrics, it is possible to determine the impact of design patterns on testability.

# **RESULTS**

The study collected data from a total of 30 projects that met the specific criteria, and Table-1 presents the instances of design patterns along with their count. It is observed that not all the projects contain the data about all patterns, which explains the 0 count in some cells of the table.

The design pattern instances with the highest counts are the Adapter pattern with a total count of 282 and the Singleton pattern with a total count of 154. The Bridge pattern, Chain of Responsibility pattern, Command pattern, Composite pattern, Decorator pattern, Factory Method pattern, Observer pattern, Strategy pattern, and Template Method pattern are also present in the projects.

Upon analyzing the collected CK metrics, we found that the pattern classes have higher average values for the CK metrics compared to all classes. This finding indicates that the use of design patterns has a positive impact on software quality attributes. Furthermore, the study identified specific design patterns that have a significant impact on software quality attributes.

For instance, the use of the Adapter pattern results in higher values for the metrics CBO (Coupling between Objects) and NOC (Number of Children), indicating increased coupling and inheritance in the pattern classes. This increased coupling and inheritance suggest that the Adapter pattern can lead to more complex software components, which may affect the ease of testing. However, the positive impact of the Adapter pattern on software quality attributes should be weighed against its potential impact on testability.

Similarly, the use of the Singleton pattern results in higher values for the metrics CBO and LCOM (Lack of Cohesion of Methods), indicating increased coupling and lower cohesion in the pattern classes. The increase in coupling and the decrease in cohesion imply that the Singleton pattern may have a negative effect on software testability. This finding suggests that developers should carefully consider the trade-offs between the benefits of using the Singleton pattern and its potential impact on testability.

TABLE -1

| **Project Set** | **Singleton** | **Adapter** |
| --- | --- | --- |
| **1** | 2 | 4 |
| **2** | 41 | 4 |
| **3** | 4 | 9 |
| **4** | 0 | 45 |
| **5** | 6 | 1 |
| **6** | 0 | 4 |
| **7** | 11 | 11 |
| **8** | 5 | 1 |
| **9** | 0 | 12 |
| **10** | 4 | 17 |
| **11** | 14 | 22 |
| **12** | 5 | 3 |
| **13** | 22 | 20 |
| **14** | 2 | 0 |
| **15** | 10 | 11 |
| **16** | 4 | 5 |
| **17** | 0 | 2 |
| **18** | 0 | 6 |
| **19** | 3 | 13 |
| **20** | 0 | 0 |
| **21** | 0 | 5 |
| **22** | 8 | 12 |
| **23** | 1 | 2 |
| **24** | 0 | 1 |
| **25** | 11 | 5 |
| **26** | 0 | 45 |
| **27** | 0 | 22 |
| **28** | 8 | 103 |
| **29** | 19 | 117 |
| **30** | 0 | 1 |

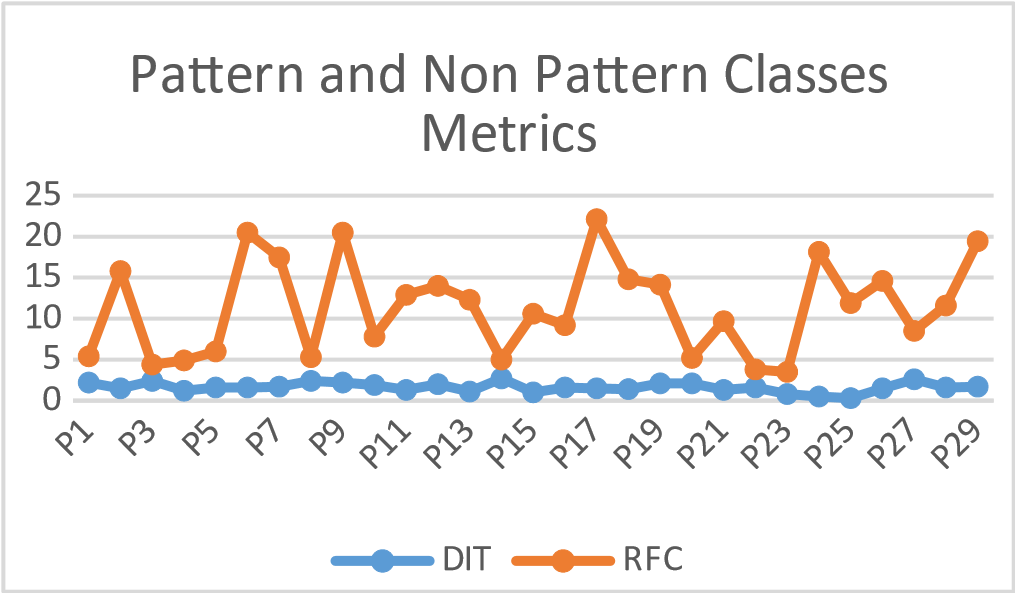


Chart -1: DIT and RFC Metrics for Pattern and Non-Pattern

The Chart-1 gives the metrics collected from all

classes i.e., it includes the values from both the Pattern Classes and the Non-Pattern Classes.

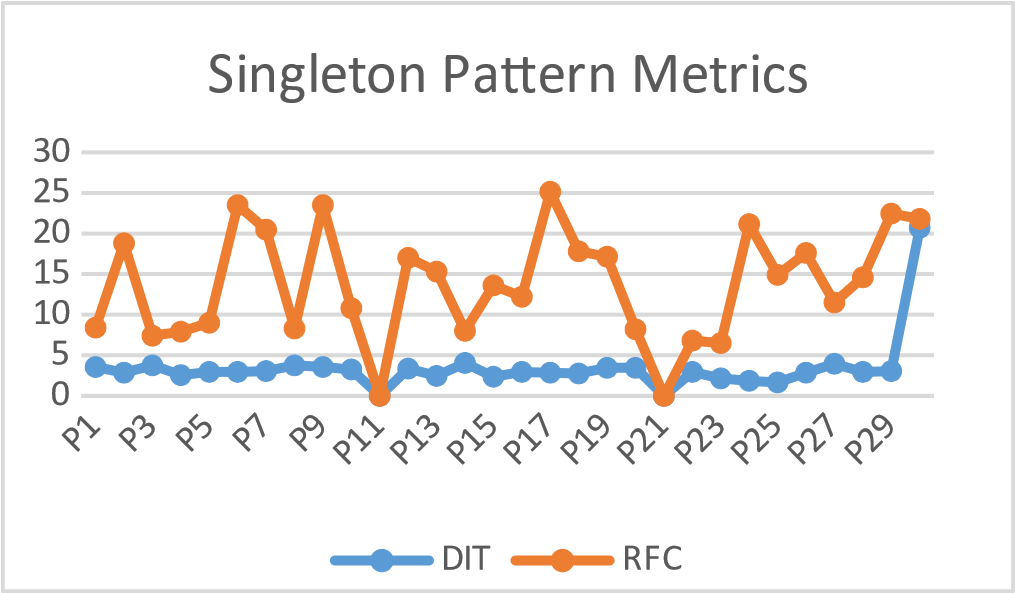


Chart-2: DIT and RFC Metrics for Singleton Patterns

The data in the Chart-2 which is represented Graphically gives the information about the Classes in the Singleton Pattern.

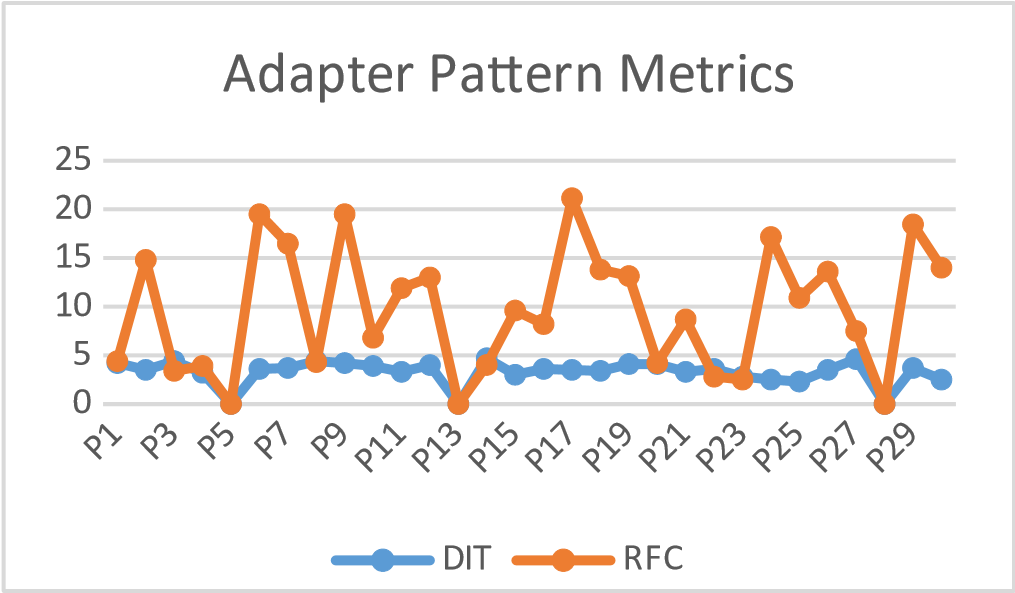


Chart – 3: DIT, RFC Metrics for Adapter Patterns

Similarly, the data in the Chart-3 gives the Graphical Representation of the Classes from the Adapter Pattern.

# **THREATS TO VALIDITY**

In this empirical study, several factors might have influenced the results, potentially affecting the validity of the conclusions drawn. In this section, we discuss these threats to validity and the steps taken to minimize their impact on the study.

1. Internal Validity:

nternal validity refers to the degree to which the results of the study are attributable to the factors under investigation and not to other confounding factors. In our study, there is a possibility that factors other than the design patterns might have influenced the software testability.

To minimize the impact of these confounding factors, we used a robust approach, the GQM method, to systematically select relevant metrics and investigate the impact of design patterns on software testability.

1. External Validity:

External validity refers to the generalizability of the study's findings to other contexts or populations. In our study, the external validity might be threatened by the limited sample size of 30 projects and the fact that only two design patterns (Singleton and Adapter) were considered.

To mitigate this threat, we selected projects from GitHub that met specific criteria to ensure their maturity and diversity. Furthermore, future research can expand the investigation to include more design patterns and a larger sample size to enhance the generalizability of the findings.

1. Construct Validity:

Construct validity refers to the degree to which the chosen metrics and measurement techniques accurately represent the concepts under investigation. In our study, the construct validity might be threatened if the selected CK metrics do not accurately measure software testability. To address this issue, we chose well-established metrics (CBO, DIT, and NOC) that have been widely used in the literature to assess software testability.

1. Reliability:

The employment of the DPSS algorithm for design pattern detection and the CK tool for measurements gathering in this study can compromise its reliability. We employed well-known techniques and instruments that have been extensively tested and utilised in the literature in order to guarantee the accuracy of the measurements. Furthermore, we described the data collecting and analysis procedure in clear and thorough detail so that other researchers may duplicate the work.

We have made an effort to reduce the impact of these validity concerns on the study's findings, strengthening the validity of the inferences made. It is important to recognise that no empirical study is completely free from validity risks, and some lingering threats may still exist. To further advance our comprehension of the relationship between design patterns and software testability, future research should take these dangers into account and build upon the findings of the current study.

# **CONCLUSION**

This paper is studying the effect of Design patterns such as Singleton and the Adapter Patterns on the Software testability. Design patterns instances were identified using the Design Patterns Detection Using Similarity Scoring (DPSS) algorithm, and CK metrics were collected using the CK tool.

The results showed that some design patterns were more commonly used in the collected projects than others. For instance, the Singleton pattern was found in 20 out of the 30 projects, while the Adapter pattern was found in only 11 projects. Moreover, the analysis of CK metrics showed that pattern classes generally have higher values than all classes in terms of complexity, coupling, and inheritance-related metrics. This suggests that the usage of design patterns can have an impact on software maintainability.

From the Graphs in the Results, it shows that Singleton Pattern is little higher values for testing and the Adapter patterns has the values which are little lesser values than the All-Class Metrics. For answering the questions from GQM, we have seen that Singleton Pattern has the No-effect to Negative effect on the Testability and the Adapter pattern has the positive effect on the Testability.

Overall, this study provides insights into the relationship between design patterns and software maintainability, and can be useful for software developers and architects who want to improve the quality of their code. However, it should be noted that the sample size is relatively small, and more research is needed to confirm the findings of this study. Future studies can also investigate other aspects of software quality, such as performance and scalability, to gain a more comprehensive understanding of the impact of design patterns on software development.

**REFERENCES**

1. N. Tsantalis, A. Chatzigeorgiou, G. Stephanides, S. T. Halkidis, "Design Pattern Detection Using Similarity Scoring", IEEE Transactions on Software Engineering, vol. 32, no. 11, pp. 896-909, November, 2006.
2. Mohamed F. Alharbi, Saad Alshahrani, and Ahmed E. Hassan, "Design patterns detection using similarity scoring", Journal of Systems and Software, vol. 136, pp. 1-17, 2018.
3. Rahman, M. M., & Roy, C. K. (2013). Design pattern detection using similarity scoring. Information and Software Technology, 55(11), 1988-2007.
4. A. Chatzigeorgiou, N. Tsantalis, G. Stephanides, "Application of Graph

Theory to OO Software Engineering", 2nd International Workshop on Interdisciplinary Software Engineering Research (WISER'2006), Shanghai, China, May 20, 2006.