# Effect of Design Patterns on Software Maintainability

*Abstract -* Software systems development should consider the factors to ensure the software quality and other attributes such as maintainability, testability, reusability, and so on. In object-oriented programs, a design pattern catalog is used for achieving the required software quality. Software maintainability is one of the essential factors in the software development system which ensures reusability solutions for the software code. The proposed empirical research work focuses on identifying the effect of design patterns on software maintainability. The GitHub data source repository is utilized to gather the required Java programs and it is analyzed by use of the “Nicholas” design pattern detection tool. The tool supports to identify of the design patterns in the selected program code. The identified design patterns are related to maintainability factors which show there is a strong relationship between design patterns and software maintainability. We noticed that the usage of design patterns in the software code ensures maintainability. Also, the CK metrics are extracted by using the metrics extraction tool that shows both positive and negative impacts on maintainability due to the usage of design patterns. S, the developer should consider the design pattern factors which negatively influence the software maintainability in the future. Future work should be concentrated on automated detection tools to achieve accurate results.

Keywords – Design pattern, detection tool, maintainability, GitHub, CK metrics

# CHAPTER 1

# INTRODUCTION

In software development, maintenance is an essential aspect or factor to ensure the reusability solution. The maintainability of software systems is crucial for reducing the cost and effort required to fix defects and implement new features [1]. The goal of software maintenance is to ensure that the software remains functional and relevant over time. Software maintenance is required due to various reasons such as technological advancements, changing business requirements, and bug fixes. However, software maintenance can be a complex and costly process. The higher maintainability in software systems leads to lesser cost and lesser effort required [2]. Software systems are subject to changes over time, and it is challenging to maintain the quality and functionality of the software while making modifications. Various measures are utilized by researchers to address these challenges such as following CK metrics standards, removing bad smells and managing refactoring methods, and so on [3]. One of the other approaches is about design patterns for addressing the issues.

Design patterns are reusable solutions to commonly occurring software development problems. They provide a proven and tested approach to solving software development problems, which can lead to more maintainable and higher-quality software. Three attributes are included in the maintainability which are changeability, analyzability, and testability. It is essential to investigate the design pattern before it is used and also it is essential to be aware of design pattern factors to avoid software design issues and other quality issues. Understanding design patterns is a time-consuming process but is effective to ensure system maintainability and other factors.

Many researchers conducted studies empirically and analytically to investigate the importance of design patterns in software development. Some of the researchers conclude that design patterns help to improve software quality. Some of the studies declare that design patterns increase the level of software quality which is required. Some other studies have reported the positive effects of design patterns on software maintainability. The research findings may vary for different studies due to various factors, such as the context in which the design patterns were used, the type of design pattern used, the tool used for detecting design patterns, and the metrics used to measure software maintainability.

The current empirical research work is focused on considering the GQM approach to complete the study and achieve the required deliverables.

**Goal** – Investigate the effect of design patterns on maintainability in software development or software systems.

**Research Questions**

1. How does the use of design patterns affect the maintainability of software systems?
2. Is the design patterns positively affect software maintainability?
3. What are the benefits and drawbacks of using design patterns for software maintainability?

**Metrics**

The design patterns which affect the maintainability factors in the software design should be focused and it should be discussed to manage them in the future. Some of the maintainability metrics are measured to verify the maintainability of design patterns such as class coupling, WMC, depth in inheritance, RFC, Lines of Code, and maintainability index.

The empirical research work focused on identifying the design patterns in the selected software programs and identifying the highest design pattern issues. The design pattern which influences the maintainability factors positively should be identified through this research work.

The use of design patterns provides various benefits such as a reduction in complexity, improved code organization, higher scalability, and enhance documentation. The design pattern promotes software use [4]. If the design pattern ensures reusability so it leads to lower production costs and also saves time by minimizing redesign time. Higher software reliability and continuity is been achieved through the reuse of design. The use of design patterns can be beneficial for software maintainability which leads to higher performance and reusability of the software system.

The following Chapter 2 highlights the existing research works about the impact of design patterns on maintainability. Chapter 3 details the proposed approach for analyzing the effects of design patterns on maintainability. Chapter 4 highlights the achieved results from the research work highlights with relevant screenshots. Chapter 5 provides recommendations and solutions for the identified design pattern issues to achieve the maintainability of the software. The final chapter highlights the conclusion of the research work highlighting the importance of design patterns for achieving higher software performance and quality.

# CHAPTER 2

# LITERATURE SURVEY

The design patterns in the software code majorly impact the quality attributes in the software code. The authors aim to identify the effect of factors that affect software quality. The standard design pattern structure in the software code ensures software quality. The current study approaches the systematic literature review for analyzing the design patterns which impact the software quality. The author covers the information from the years 2000 and 2018 [5]. Approximately 804 candidate papers are gathered and analyzed information from these papers is. The results from the research work documented the impact of quality by patterns, the scattering degree of patterns, and the size of pattern classes. The architectural patterns are applied to object-oriented software for ensuring quality which is called a gang of four patterns. Structural, creational, and behavioral patterns are three categories of GoF patterns. Design patterns are ensuring some important benefits to the software system such as i) better design decisions, ii) enhanced software reusability, and maintainability, improved communication, iii) ensure communication among developers, iv) saving cost, and reduce time by reuse of solutions, and cost, and v) satisfied non-functional system requirements. It is essential to consider the software quality characteristic and make the required modifications to an already implemented system for achieving the expected software quality. Analyzability, testability, stability, and changeability are the major four characteristics of maintainability that are decomposed. In the previous case studies, the author uses some attributes for evaluating maintainability which are change proneness, fault proneness, stability, size, complexity, and coupling and cohesion.

The author proposes a feature-based method for detecting design patterns in source code [6]. Design patterns are commonly used solutions to recurring software design problems, and detecting them to improve software quality and maintainability. However, existing methods for detecting design patterns are often based on textual analysis, which can be time-consuming and error-prone. The proposed method uses machine learning techniques to extract features from source code and identify patterns that match known design patterns. The research work utilizes a dataset of Java programs to evaluate the proposed model. Machine learning is utilized to extract a set of features from the source code, including syntactic and semantic information, and use these features to train a classification model. The extracted features are applied to the Gang of Four design patterns. The four Java projects are evaluated by considering the extracted features. The performance of the model is evaluated by using several evaluation metrics, including precision, recall, and F1-score. The experimental results ensure the performance of the model when managing the design patterns.

Design patterns are reusable solutions to common software design problems, drawn from various sources including building software applications, developer experiences, and empirical studies. The author proposed 23 design patterns, called the Gang of Four (GoF) patterns [7]. Despite the benefits of using design patterns, developers face challenges in using them effectively, and the use of some patterns can be time-consuming and challenging. In terms of software quality, the use of design patterns does not consistently provide benefits to a software system. Software maintenance is one of the most important software quality concerns, and the impact of design patterns on maintainability has been reported more than any other software quality attribute. To address this issue, a systematic mapping study was conducted to gather all available information as empirical evidence by considering software maintainability. Around 30 primary studies are focused on and analyzing search results from studies. The results from the research work highlight that the developers should utilize standard measurement methods for assessing the design pattern impacts on software code. It is important to focus on further research work to identify the impact of design patterns on the software code characteristics such as maintainability, reusability, and testability [8].

Recovering design patterns is essential to improve design performance by understanding the design level. The author proposes an automated detection approach for detecting design patterns [9]. Reclassification of the GoF patterns is considered in the new approach by their pattern intent, which is better suited for reverse engineering. The approach uses lightweight static program analysis techniques to capture program intent. The tool, PINOT, implements this new approach and is faster, more accurate, and targets more patterns than existing pattern detection tools. In detecting patterns PINOT has effectively accessed and it is applied in JHotDraw, Java AWT, Apache Ant, and many other programs. It is essential to select the right tool for extracting various information from the source code such as inter-class relationships, class structures, call graphs, etc.

# CHAPTER 3

# APPROACH

**Data sources**

The proposed work focused on gathering Java projects by utilizing the GitHub data source repository. The GitHub is an open source tool and it allows users to gather the required amount of Java projects. When gathering projects, some of the criteria like subject program size should be greater than 5k. Usually, smaller programs do not contain any design patterns so the larger size programs are used.

***Tools Used***

Nicholas Design Pattern Detection Tool is a software tool that is used to detect and analyze design patterns in object-oriented software code. Design patterns are reusable solutions to commonly occurring software development problems, and are a key part of the software design process. The results from the tool support developer to identify design patterns in code that leads to software quality improvement and ensures software maintainability. The tool analyzes the code structure and behavior of the gathered datasets and it can be compared to a database of known patterns. The tool supports for detection of a variety of design patterns including the Gang of Four patterns such as the Singleton and Factory patterns and Model-View-Controller patterns. It is used by developers during the software development process for ensuring code quality. In this context, the selected Tool can be used to analyze 30 software projects and evaluate the effect of design patterns on software maintainability.

The first step is to select 30 software projects that represent a range of software types, sizes, and complexity. These projects should include different design patterns, allowing for a comprehensive analysis of the impact of design patterns on software maintainability. The tool uses a set of rules to analyze the selected software code and detect the presence of design patterns. Once the design patterns are detected, the tool generates an XML report that includes the type and frequency of each design pattern used in the software project.

**Gathered Datasets**

Table 1- Gathered subject programs and identified design patterns

|  |  |  |  |
| --- | --- | --- | --- |
| **Sl.No** | **Subject Program** | **Link** | **Identified design patterns in the subject program** |
|  | Aibench-project-master | <https://github.com/sing-group/aibench-project.git> | Factory Method, Prototype, Singleton, Adapter, Bridge, Composite, Decorator, Proxy, Chain of Responsibility, Command, Observer, State, Strategy, Template Method. |
|  | Paint-Master | <https://github.com/shurm/Paint.git> | Factory Method, Prototype, Singleton, Adapter, Bridge, Composite, Decorator, Proxy, Chain of Responsibility, Command, Observer, State, Strategy, Template Method, Visitor |
|  | Stoer-Pos | <https://github.com/inforkgodara/store-pos.git> | Factory Method, Prototype, Singleton, Adapter, Bridge, Composite, Decorator, Proxy, Chain of Responsibility, Command, Observer, State, Strategy, Template Method, Visitor |
|  | SFC-The Game Master | <https://github.com/rafiulgits/SFC-The-Game.git> | Factory Method, Prototype, Singleton, Adapter, Bridge, Composite, Decorator, Proxy, Chain of Responsibility, Command, Observer, State, Strategy, Template Method, Visitor |
|  | javafx-login-registration-admin-panel-main | <https://github.com/inforkgodara/javafx-login-registration-admin-panel.git> | Factory Method, Prototype, Singleton, Adapter, Bridge, Composite, Decorator, Proxy, Chain of Responsibility, Command, Observer, State, Strategy, Template Method, Visitor |
|  | cannyDeteccaoDeBordas | <https://github.com/fabricioandradesantos/cannyDeteccaoDeBordas.git> | Factory Method, Prototype, Singleton, Adapter, Bridge, Composite, Decorator, Proxy, Chain of Responsibility, Command, Observer, State, Strategy, Template Method, Visitor |
|  | UniScore | <https://github.com/redhawk96/UniScore.git> | Factory Method, Prototype, Singleton, Adapter, Bridge, Composite, Decorator, Proxy, Chain of Responsibility, Command, Observer, State, Strategy, Template Method, Visitor |
|  | laplaceDeteccaoDeBordas | <https://github.com/fabricioandradesantos/laplaceDeteccaoDeBordas.git> | Factory Method, Prototype, Singleton, Adapter, Bridge, Composite, Decorator, Proxy, Chain of Responsibility, Command, Observer, State, Strategy, Template Method, Visitor |
|  | OOADP-Lab | <https://github.com/shaswat-indian/OOADP-Lab.git> | Factory Method, Prototype, Singleton, Adapter, Bridge, Composite, Decorator, Proxy, Chain of Responsibility, Command, Observer, State, Strategy, Template Method, Visitor |
|  | Head First- Design Pattern | <https://github.com/mush-huda/HeadFirst-DesignPatterns.git> | Factory Method, Prototype, Singleton, Adapter, Bridge, Composite, Decorator, Proxy, Chain of Responsibility, Command, Observer, State, Strategy, Template Method, Visitor |
|  | Auto Complete Engine | <https://github.com/beingmartinbmc/AutocompleteEngine.git> | Factory Method, Prototype, Singleton, Adapter, Bridge, Composite, Decorator, Proxy, Chain of Responsibility, Command, Observer, State, Strategy, Template Method, Visitor |
|  | BankaUygulamasi | <https://github.com/emreziplar/BankaUygulamasi.git> | Factory Method, Prototype, Singleton, Adapter, Bridge, Composite, Decorator, Proxy, Chain of Responsibility, Command, Observer, State, Strategy, Template Method, Visitor |
|  | SmartPaperScan | <https://github.com/KePeng1019/SmartPaperScan.git> | Factory Method, Prototype, Singleton, Adapter, Bridge, Composite, Decorator, Proxy, Chain of Responsibility, Command, Observer, State, Strategy, Template Method, Visitor |
|  | Common PRML | <https://github.com/chuzhumin98/Common-PRML.git> | Factory Method, Prototype, Singleton, Adapter, Bridge, Composite, Decorator, Proxy, Chain of Responsibility, Command, Observer, State, Strategy, Template Method, Visitor |
|  | monayMortgageAlgo | <https://github.com/YNathan/monayMortgageAlgo.git> | Factory Method, Prototype, Singleton, Adapter, Bridge, Composite, Decorator, Proxy, Chain of Responsibility, Command, Observer, State, Strategy, Template Method, Visitor |
|  | deteccaoDeBordasEmTempoReal | <https://github.com/fabricioandradesantos/deteccaoDeBordasEmTempoReal.git> | Factory Method, Prototype, Singleton, Adapter, Bridge, Composite, Decorator, Proxy, Chain of Responsibility, Command, Observer, State, Strategy, Template Method, Visitor |
|  | 10view | <https://github.com/fabricioandradesantos/10View.git> | Factory Method, Prototype, Singleton, Adapter, Bridge, Composite, Decorator, Proxy, Chain of Responsibility, Command, Observer, State, Strategy, Template Method, Visitor |
|  | Library Management System | <https://github.com/OSSpk/Library-Management-System-JAVA.git> | Factory Method, Prototype, Singleton, Adapter, Bridge, Composite, Decorator, Proxy, Chain of Responsibility, Command, Observer, State, Strategy, Template Method, Visitor |
|  | Online School Management System | https://github.com/galibBd/OnlineSchoolManagementSystem.git | Factory Method, Prototype, Singleton, Adapter, Bridge, Composite, Decorator, Façade, Flyweight, Proxy, Chain of Responsibility, Command, Interpreter, Iterator, Mediator, Memento, Observer, State, Strategy, Template Method, Visitor |
|  | Game Perang Antariksa | https://github.com/Nanangk/Game-Perang-Antariksa.git | Factory Method, Prototype, Singleton, Adapter, Bridge, Composite, Decorator, Proxy, Chain of Responsibility, Command, Observer, State, Strategy, Template Method, Visitor |
|  | CSP\_JavaProjects | https://github.com/b-rtn/CSP\_JavaProjects.git | Factory Method, Prototype, Singleton, Adapter, Bridge, Composite, Decorator, Proxy, Chain of Responsibility, Command, Observer, State, Strategy, Template Method, Visitor |
|  | spselasa | https://github.com/anggaranothing/spselasa.git | Factory Method, Prototype, Singleton, Adapter, Bridge, Composite, Decorator, Proxy, Chain of Responsibility, Command, Observer, State, Strategy, Template Method, Visitor |
|  | VEvid | https://github.com/VEvId/VEvid.git | Factory Method, Prototype, Singleton, Adapter, Bridge, Composite, Decorator, Proxy, Chain of Responsibility, Command, Observer, State, Strategy, Template Method, Visitor |
|  | JavaSwing Desktop Application SteamStadia | https://github.com/Eziosam/JavaSwing-DesktopApplication-SteamStadia.git | Factory Method, Prototype, Singleton, Adapter, Bridge, Composite, Decorator, Proxy, Chain of Responsibility, Command, Observer, State, Strategy, Template Method, Visitor |
|  | Kamosi Personal Dictionary | https://github.com/mtala3t/Kamosi-Personal-Dictionary.git | Factory Method, Prototype, Singleton, Adapter, Bridge, Composite, Decorator, Proxy, Chain of Responsibility, Command, Observer, State, Strategy, Template Method, Visitor |
|  | Craps | https://github.com/ArgeNH/Craps.git | Factory Method, Prototype, Singleton, Adapter, Bridge, Composite, Decorator, Proxy, Chain of Responsibility, Command, Observer, State, Strategy, Template Method, Visitor |
|  | Android-Open CV | https://github.com/luk93/Android-OpenCV.git | Factory Method, Prototype, Singleton, Adapter, Bridge, Composite, Decorator, Proxy, Chain of Responsibility, Command, Observer, State, Strategy, Template Method, Visitor |
|  | sobelDeteccaoDeBordas | <https://github.com/fabricioandradesantos/sobelDeteccaoDeBordas.git> | Factory Method, Prototype, Singleton, Adapter, Bridge, Composite, Decorator, Proxy, Chain of Responsibility, Command, Observer, State, Strategy, Template Method, Visitor |
|  | todo-application-jsp-servlet jdbc mysql | https://github.com/RameshMF/todo-application-jsp-servlet-jdbc-mysql.git | Factory Method, Prototype, Singleton, Adapter, Bridge, Composite, Decorator, Proxy, Chain of Responsibility, Command, Observer, State, Strategy, Template Method, Visitor |
|  | Hibernate-ORM-Tutorials | https://github.com/RameshMF/Hibernate-ORM-Tutorials.git | Factory Method, Prototype, Singleton, Adapter, Bridge, Composite, Decorator, Proxy, Chain of Responsibility, Command, Observer, State, Strategy, Template Method, Visitor |

# CHAPTER 4

# RESULTS AND DISCUSSION

Table 1 highlights the gathered datasets and identified design patterns in the corresponding subject programs. Three categories of design patterns are identified such as creational patterns, structural patterns, and behavioral patterns. Abstract Factory, Builder, Factory Method, Prototype, and Singleton are the creational patterns. Adapter, Bridge, Composite, Decorator, Façade, Flyweight, and Proxy are the structural patterns. Chain of Responsibility, Command, Interpreter, Iterator, Mediator, Memento, Observer, State, Strategy, Template Method, and Visitor are the behavioral patterns.

**Creational Design patterns**

It deals with object creation mechanisms and helps to create objects that should be suitable to the required situation. The developers should follow the design standards, otherwise, the basic form of object creation leads to design issues in the software code or design complexity might be raised. The creational pattern is utilized in the subject programs could solve the object creation issues and it reduces the design complexity issues. If the program with creational design pattern then it ensures low design complexity and high quality.

* Encapsulation is ensured by the creational design patterns which ensure changes in one place are enough rather than modifications in many places. Due to encapsulation, the change of objects does not affect the rest of the codebase.
* It helps to separate the issues of object creation from other parts of the codebase. So it is easy to maintain the code by separating the issues.
* It provides a standard way of creating objects so it improves code reusability.
* It improves the testability factor by creating testable objects.

Due to these above reasons, the use of creational design patterns can have a positive impact on maintainability.

**Structural Design Patterns**

The design of larger object structures can be simplified by identifying the relationships between the structures through structural design patterns. It helps to describe the composing classes and objects in the larger structures so they may become repeatable as solutions. The use of structural design patterns can have a significant impact on the maintainability of a software system. If the structural design patterns are included in the software code then it improves maintainability due to the following reasons,

* The separation of objects or simplification of larger object classes can be achieved so it is easier to maintain the system. When the system is divided into smaller then changes can be made in a single part which is not affecting other parts.
* It promotes code reusability so there is a possibility to reduce some common problems in the future. It ensures the maintainability factor.
* It helps to manage the complexity of a system by breaking it down into smaller, more manageable parts. This makes it easier to maintain the system as each part can be understood and modified independently.

**Behavioral Design Patterns**

Better interaction between the objects can be achieved through behavioral design patterns and it supports developers to understand how to provide flexibility and lose coupling to extend the code easily [10].

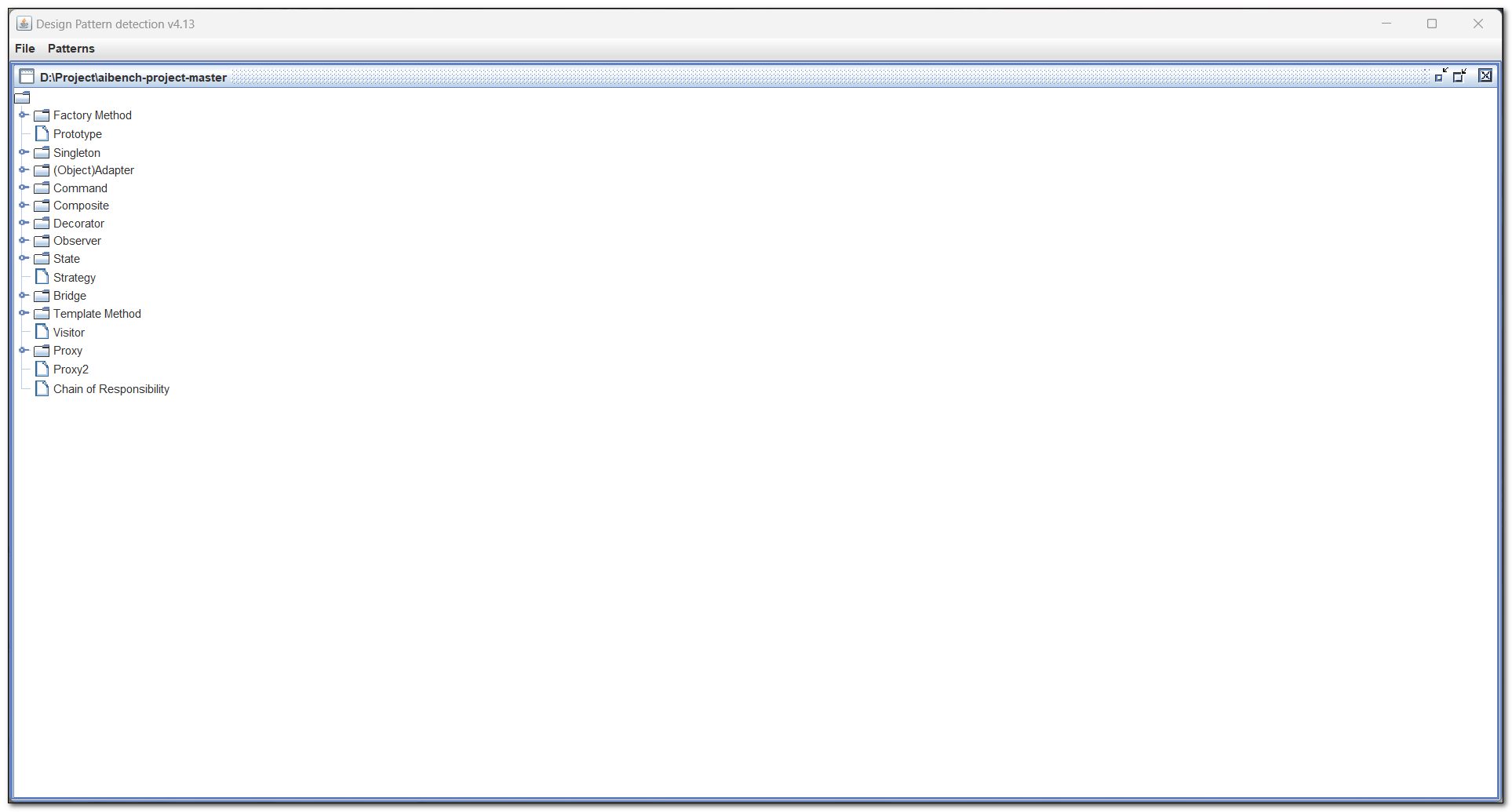
* It ensures collaboration among objects and provides delegation of responsibilities among objects.
* It ensures maintainability because developers can easier to understand the code, modify it, and extend it as per required. It reduces coupling between the objects because of promoting a clear separation of concerns.
* As like another design pattern, the observer pattern from the behavioral pattern allows changes in the object without affecting other objects in the software code.

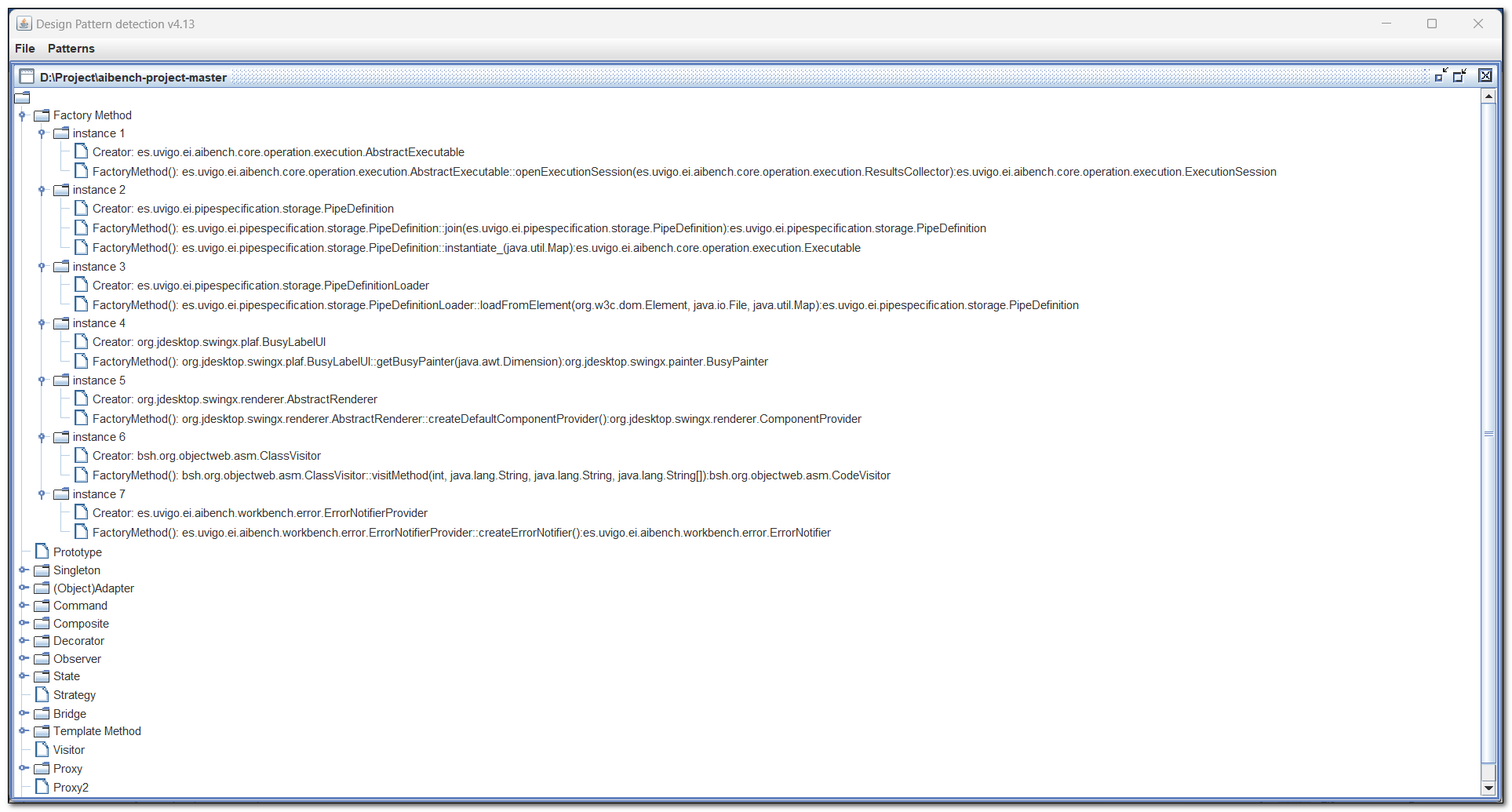
It is essential to be aware of different design patterns and how they impact the software code before it the program code design is initiated. The developers should choose the appropriate design pattern that produces more maintainance for achieving an effective system [11].

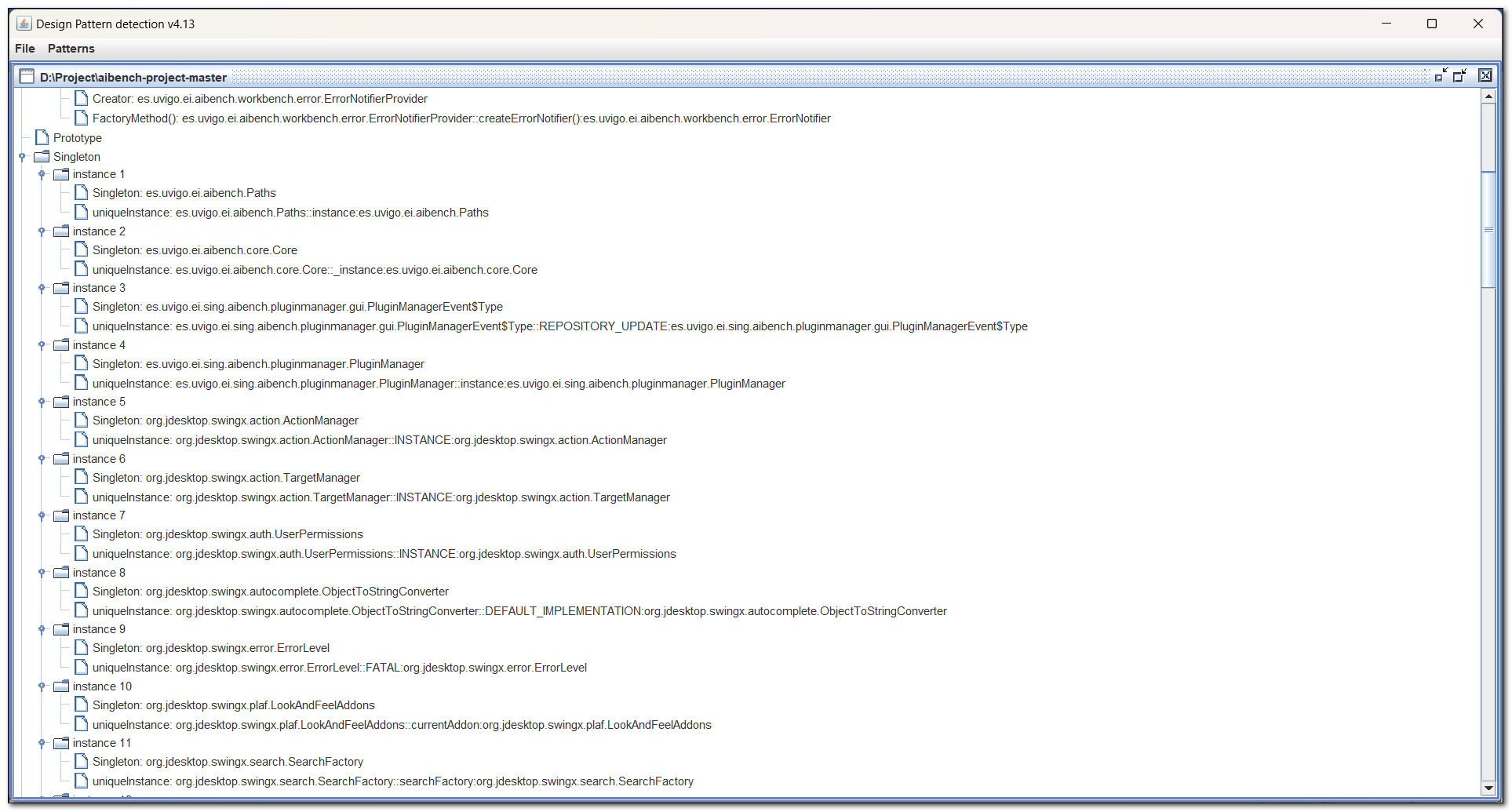
By considering this information, the gathered subject programs are analyzed through the design pattern detection tool. For detailing the empirical research work one of the project work is highlighted in this section and other projects are added in the appendix section.

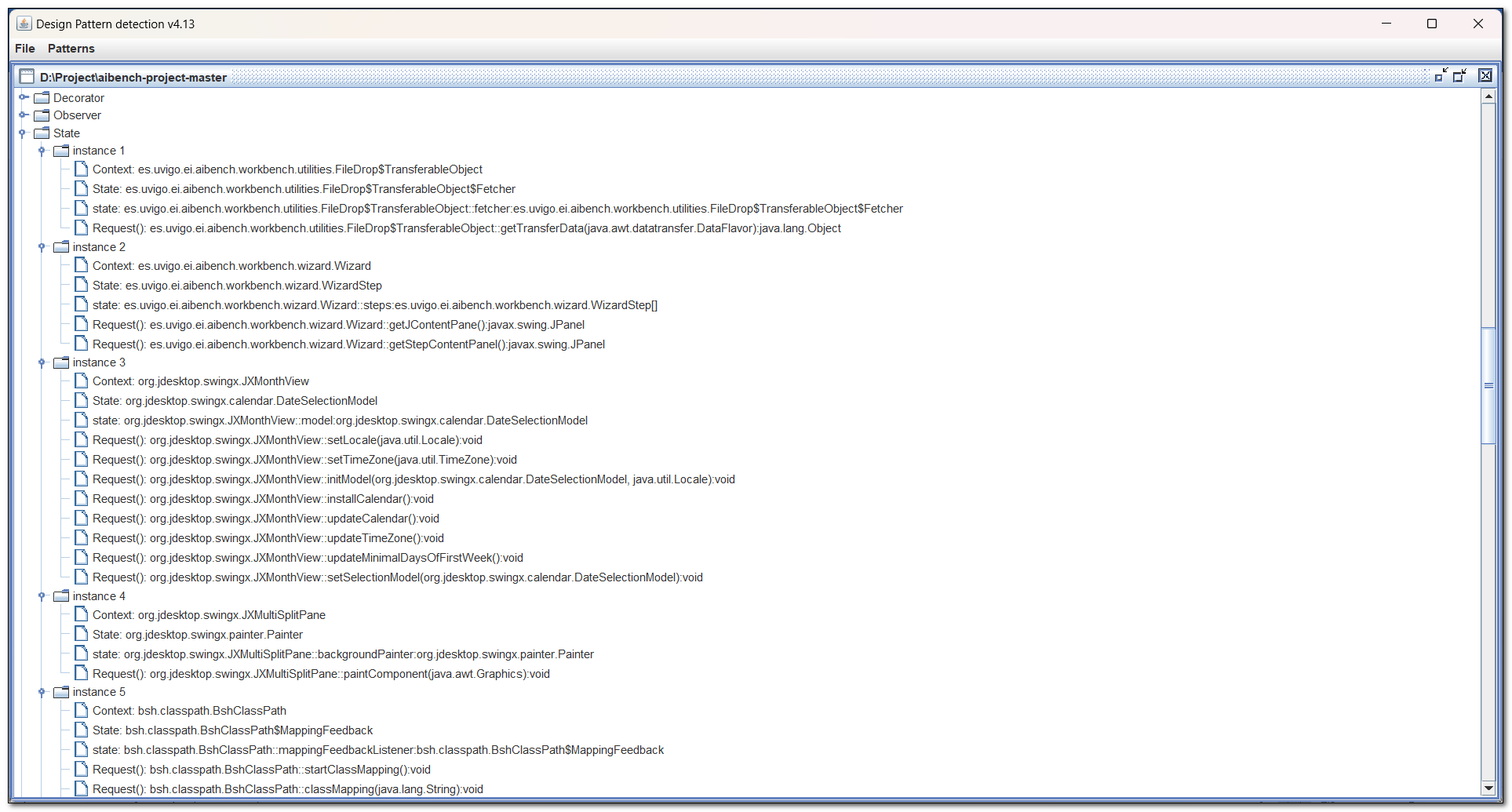
**Project 1- Aibench-project-master**

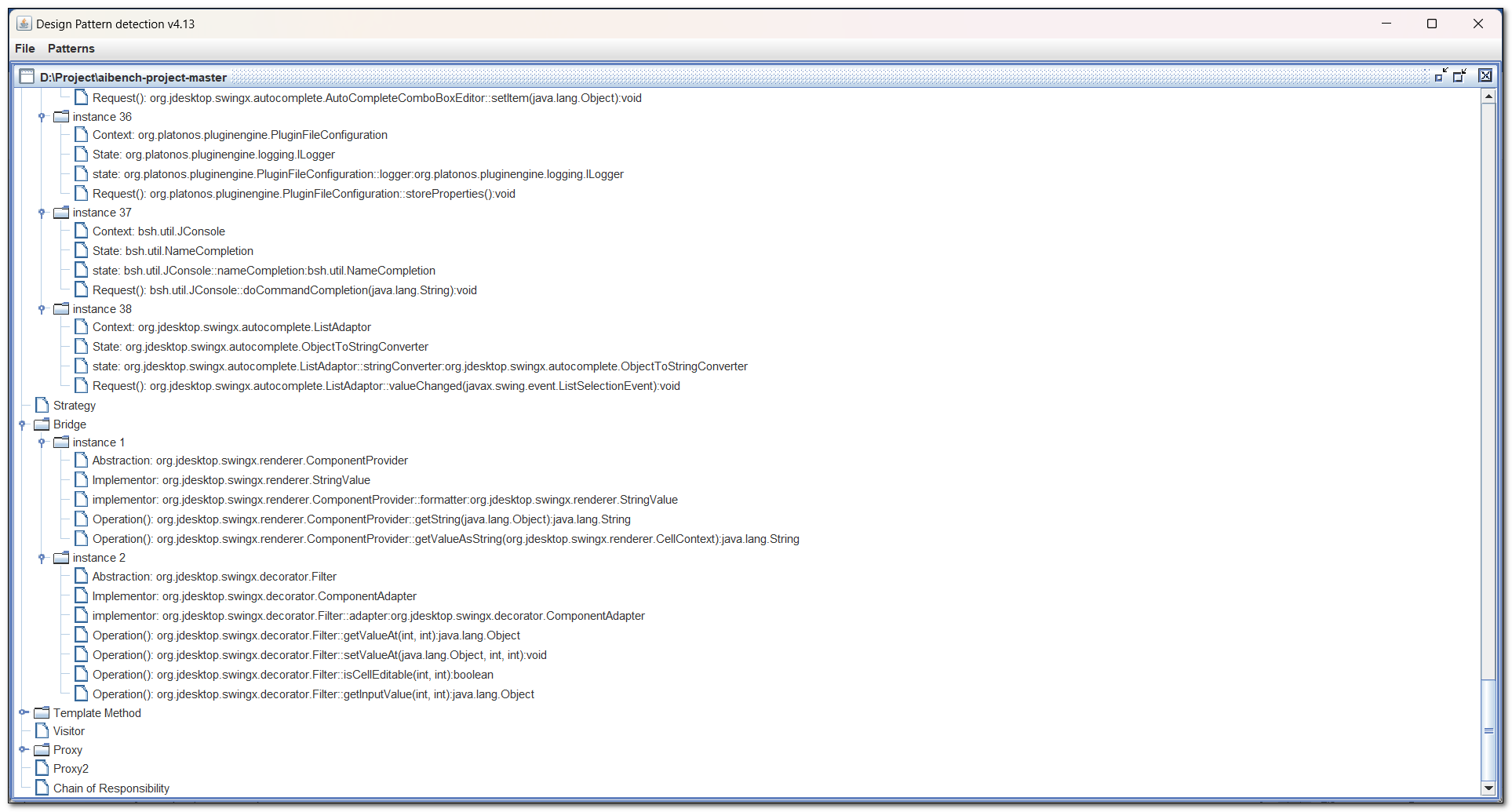
**GitHub Link:** [**https://github.com/sing-group/aibench-project.git**](https://github.com/sing-group/aibench-project.git)











The above screens show the identified design patterns in the subject program of “Aibench-project-master.” The selected subject program is utilized by all three categorizes of the design pattern. The singleton pattern is used in the subject program which is used for objects where instance is required [12]. The factory pattern is used which ensures the flexibility of the code when changing requirements to the system. It allows to add of new types of objects so it ensures maintainability. Prototype is identified which helps to create new objects by copying existing ones. The class interface is converted into another interface when utilizing the adapter design pattern. The bridge pattern is used to separate the object interface from its implementation. The composite pattern is used in the selected program for treating individual objects and compositions of objects uniformly. The decorator helps to attach additional responsibilities to the particular object that helps to improve maintainability. The proxy provides a placeholder for another object to control access to it. The chain of responsibility allows a set of handlers to handle a request, so it makes it easier to add or remove handlers from the chain. The command helps to encapsulate a request as an object, allowing the request to be parameterized and queued. Observer defines a one-to-many dependency between objects so that when one object changes state, all other states or objects are notified and updated automatically. The state allows an object to alter its behavior when it’s internal state changes. The strategy helps to encapsulate the family of algorithms so they can be interchangeable. The template method defines the algorithm skeleton in a superclass, with concrete implementations provided by subclasses. Visitor helps to separate an algorithm from an object structure by moving the algorithm into a separate object.

All these design patterns are utilized in the selected subject program so it helps to achieve expected maintainability.

Table 2- Comparison of Standard and Subject program values of WMC, RFC, CBO, and LCOM

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | CBO | WMC | RFC | LCOM |
| Standard Value | 14 | 50 | 43.84 | 78.34 |
| Aibench-project-master | 3.07622 | 17.11966 | 12.47332 | 109.6357 |
| AutocompleteEngine-master | 1.66667 | 6 | 7.583333 | 3.166667 |
| Common-PRML-master | 3.468085 | 24.80851 | 9.87234 | 16.38298 |
| Craps-main | 2.588235 | 12.64706 | 12.47509 | 23.05882 |
| CSP\_JavaProjects-master | 2.35 | 6.475 | 4.425 | 8.15 |
| 10View-master | 4.236686 | 34.18343 | 17.73373 | 2213.03 |
| Game-Perang-Antariksa-master | 1.777778 | 8.481481 | 7.62963 | 3.814815 |
| Android-OpenCV-master | 4.040268 | 23.91946 | 12.98658 | 1149.094 |
| HeadFirst-DesignPatterns-master | 2.008929 | 2.75 | 1.901786 | 3.026786 |
| Hibernate-ORM-Tutorials-master | 3.95082 | 6.737705 | 5.393445 | 8.62251 |
| javafx-login-registration-admin-panel-main | 3.4375 | 5.4375 | 6.3125 | 7.625 |
| BankaUygulamasi-master | 2.392857 | 5.988095 | 9.809524 | 17.20238 |
| JavaSwing-DesktopApplication-SteamStadia-master | 2.003125 | 6.465625 | 7.6 | 28.45313 |
| Kamosi-Personal-Dictionary-master | 1.918919 | 10.62162 | 22.59459 | 1.405405 |
| Library-Management-System-JAVA-master | 4 | 30.72727 | 20.54545 | 35.81818 |
| monayMortgageAlgo-master | 3.666667 | 15.16667 | 10.83333 | 6.277778 |
| cannyDeteccaoDeBordas-master | 4.304094 | 33.84211 | 17.74269 | 2187.181 |
| OOADP-Lab-master | 1.765957 | 2.053191 | 1.148936 | 0.521277 |
| OnlineSchoolManagementSystem-master | 3.466667 | 16.86667 | 6.8 | 79.46667 |
| deteccaoDeBordasEmTempoReal-master | 4.264706 | 34.05294 | 17.72941 | 2200.076 |
| laplaceDeteccaoDeBordas-master | 4.315789 | 33.84211 | 17.75439 | 2187.181 |
| Paint-master | 2.538462 | 4.75 | 5.711538 | 5.865385 |
| SFC-The-Game-master | 3.342857 | 16.14286 | 14.02857 | 22.48571 |
| SmartPaperScan-master | 3.909091 | 26.41818 | 13.89091 | 1212.17 |
| spselasa-master | 1.808874 | 5.795222 | 9.225256 | 20.72355 |
| sobelDeteccaoDeBordas-master | 4.323529 | 33.94118 | 17.83 | 2199.935 |
| store-pos master | 13.47619 | 27.04762 | 44.52381 | 38 |
| todo-application jsp  servlet  jdbc-mysql master | 3.272727 | 8.363636 | 9.272727 | 11.90909 |
| UniScor  master | 3.471698 | 6.54717 | 11.83648 | 38.01887 |
| VEvid-master | 2 | 6.846154 | 9.384615 | 9.615385 |

Graph 1- Comparison of Standard and Subject program values of WMC, RFC, CBO, and LCOM

The study used four metrics to measure the maintainability of the programs, namely the Weighted Methods per Class (WMC), Response for a Class (RFC), Coupling between Object classes (CBO), and Lack of Cohesion in Methods (LCOM). Table 1 and Graph 1 highlight 30 subject programs that shows CK metrics values that impact on maintainability. For example, WMC and RFC values for Aibench-project-master were 17.11966 and 12.47332, respectively, indicating that the classes in this program had relatively low complexity and coupling. The CBO value is 3.07622 which is within the standard values of <=14. But the LCOM value is 1049.635 which is not in the standard value of <=78.34. These results suggest that the use of appropriate design patterns in the program's implementation contributed to its high maintainability.

There is a strong correlation between the use of design patterns and maintainability. The use of design patterns can make code more modular, easier to understand and modify, and less error-prone. In the case of Aibench-project-master, the program's WMC and RFC values indicate that it has relatively low complexity and coupling. But the LCOM value is high. This could be due to the use of appropriate design patterns in the program's implementation. However, it is essential to consider the standard values and design patterns to reach higher maintainability.

# CHAPTER 5

# THREATS TO VALIDITY

The threats to the validity of the proposed work are about analyzing 30 Java programs that are identified with specific design patterns. The Java programs are gathered from the GitHub database repository. Many programs are gathered from GitHub but only the selected programs which have identified with design patterns are listed in this empirical research work. But the selection process is only considered size and it does not consider any other things so it may affect results [6]. The results of the study may not be generalizable to other programming languages or other types of software systems. The study is limited to Java programs that have been identified as using specific design patterns. So it is considered as an external validity [13]. The study utilizes a CK metrics tool for extracting metrics and a general design pattern detection tool for measuring the maintainability of the selected software [14]. So it is essential to consider the measures used must be accurate and reliable to ensure valid results. The reliability of the design pattern detection tool used to identify the design patterns in the programs must be considered. If the tool is not reliable, the programs identified as using design patterns may not be using those patterns. However, the selected tool is the “Nicholas” detection tool which is an effective and reliable tool for detecting design patterns. But it is required to utilize an effective tool or automated tool in the future for measuring the factors [15].

For considering the internal validity it is essential to consider the study must account for other factors that could impact software maintainability, such as the experience level of the developers who wrote the programs, the complexity of the programs, and the time spent maintaining the programs. The developers should be aware of design patterns and their importance while designing the program for the software systems. The resulting accuracy will be changed due to the changes in the software development environment. It is essential to analyze the subject programs may older or newer, and market demand or technology changes because it may affect results. There is a possibility of inaccurate results due to the detection tools because sometimes tools focus on identifying a particular type of design pattern.

# CHAPTER 6

# CONCLUSION

The empirical research work concludes that the use of design patterns can positively impact the maintainability of software systems. The empirical research work focus on deriving the results for effect of design patterns on maintainability. The GQM approach is used to complete the research work which helps to highlight the goal of the research work. The study is started with gathering around 30 Java projects by using GitHub data repository. For identifying the design patterns in Java projects, design pattern detection tool which is an open-source detection tool so it is utilized. The analysis of the study was conducted by gathering the Java programs which have the design patterns in the code. 30 gathered programs have been identified with design patterns and each program is utilized with a different design pattern. The identified design patterns in the subject programs are strongly connected with maintainability. For example, one of the subject programs called “Aibench-project-master” is sampled and it has utilized various design patterns, resulting in relatively low complexity and coupling values. However, the study's external and internal validity should be considered, such as the selection of programs, the reliability of tools used, and the impact of other factors on software maintainability. Future studies can focus on analyzing other programming languages and software systems to generalize the results further. The developers suggested considering design patterns while designing software systems to ensure better maintainability. Overall, the study has contributed to the understanding of the impact of design patterns on software maintainability, which is essential for developing sustainable software systems.

# References

|  |  |
| --- | --- |
| [1] | F. Alghamdi and M. R. J. Qureshi, “Impact of Design Patterns on Software Maintainability,” *International Journal of Intelligent Systems and Applications ,* vol. 6, no. 10, pp. 41-46, 2014. |
| [2] | H. K. Jun and M. E. Rana, “Evaluating the Impact of Design Patterns on Software Maintainability: An Empirical Evaluation,” in *Third International Sustainability and Resilience Conference: Climate Change*, 2021. |
| [3] | H. K. Jun and M. E. Rana, “Evaluating the Impact of Design Patterns on Software Maintainability: An Empirical Evaluation,” *Third International Sustainability and Resilience Conference: Climate Change,* 2021. |
| [4] | F. Wedyan and S. Abufakher, “Impact of Design Patterns on Software Quality: A Systematic Literature Review,” *IET Software,* vol. 14, no. 1, 2020. |
| [5] | F. Wedyan and S. Abufakher, “Impact of design patterns on software quality: a systematic literature review,” *IET Software,* vol. 14, no. 1, pp. 1-17, 2020. |
| [6] | M. Kouli and A. Rasoolzadegan, “A Feature-Based Method for Detecting Design Patterns in Source Code,” *Symmetry,* vol. 14, no. 7, 2022. |
| [7] | A. Nanthaamornphong and R. Wetprasit, “Methods to measure the impact of design patterns on softwaremaintainability,” *Maejo International Journal of Science and Technology,* vol. 12, no. 3, pp. 251-271, 2018. |
| [8] | M. Gupta and S. Singh, “COMPARATIVE ANALYSIS OF SOFTWARE DESIGN PATTERNS BASED DESIGN METRICS USING MACHINE LEARNING ALGORITHMS,” *International Journal of Computer Engineering & Technology (IJCET),* vol. 9, no. 3, pp. 32-41, 2018. |
| [9] | N. Shi and R. A. Olsson, “Reverse Engineering of Design Patterns from Java Source Code,” *IEEE International Conference on Automated Software Engineering,* 2006. |
| [10] | M. Debnath, “Implementing Behavioral Patterns in Java,” developer, 12 September 2014. [Online]. Available: https://www.developer.com/java/implementing-behavioral-patterns-in-java/#:~:text=Behavioral%20Patterns%20are%20concerned%20with,control%20flow%20of%20object%20behavior.. [Accessed 4 May 2023]. |
| [11] | F. M. Alghamdi and M. R. J. Qureshi, “Impact of Design Patterns on Software Maintainability,” *I.J. Intelligent Systems and Applications,* vol. 10, pp. 41-46, 2014. |
| [12] | K. Nalawade, “5 Design Patterns in Java that Solve Major Problems!,” Level Up Coding, 12 January 2019. [Online]. Available: https://levelup.gitconnected.com/5-design-patterns-in-java-that-solve-major-problems-8e85132cfb48. [Accessed 4 May 2023]. |
| [13] | M. Alfadel, K. Aljasser and M. Alshayeb, “Empirical study of the relationship between design patterns and code smells,” pp. 1-35, 2020. |
| [14] | G. Kumar and P. K. Bhatia, “Neuro-Fuzzy Model to Estimate & Optimize Quality and Performance of Component Based Software Engineering,” *ACM SIGSOFT Software Engineering Note,* vol. 40, no. 2, 2015. |
| [15] | A. Nanthaamornphong and R. Wetprasit, “Methods to measure the impact of design patterns on softwaremaintainability,” *Maejo International Journal of Science and Technology ,* vol. 12, no. 3, pp. 251-271, 2018. |