

**A COMPONENT BASED USER INTERACTIVE DESIGN
PATTERN RECOMMENDATION TOOL**

Udagamage Don Nilani Damayanthika Gunasekara

179319X

M.Sc. in Computer Science

Department of Computer Science Engineering

University of Moratuwa

Sri Lanka

May 2019

A COMPONENT BASED USER INTERACTIVE DESIGN PATTERN RECOMMENDATION TOOL

Udagamage Don Nilani Damayanthika Gunasekara

179319X

This dissertation submitted in partial fulfillment of the requirements for the Degree
of MSc in Computer Science specializing in Software Architecture

Department of Computer Science Engineering

University of Moratuwa

Sri Lanka

May 2019

DECLARATION

I declare that this is my own work and this dissertation does not incorporate without acknowledgement any material previously submitted for degree or Diploma in any other University or institute of higher learning and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text.

Also, I hereby grant to University of Moratuwa the non-exclusive right to reproduce and distribute my dissertation, in whole or in part in print, electronic or other medium. I retain the right to use this content in whole or part in future works (such as articles or books).

Signature:

Date:.....

Name: U.D.N.D. Gunasekara

The supervisor/s should certify the thesis/dissertation with the following declaration.

I certify that the declaration above by the candidate is true to the best of my knowledge and that this report is acceptable for evaluation for the CS5999 PG Diploma Project.

Signature of the supervisor:

Date:

Name: Dr. Indika Perera

Abstract

In today's context, growth of software industry is very rapid and the complexity of the software systems is increasingly high. To cope with the growing complexity, enhancement in the existing system is required. Design patterns offer effective ways of developing high quality products by providing best practices, design knowledge and reusable implementations. For a novice developer it is a hard task to select a proper design pattern to the knowledge he has. There are research studies carried out to suggest design patterns for a given problem scenario, but they are not focused on how the design pattern is to be selected. In this paper the researcher proposes a user interactive component based design pattern recommendation tool, to learn concepts behind selecting and suggesting design patterns for a given problem. A proof of concept is developed to evaluate the suggested tool which supports 23 design patterns described by the Gang of Four (GoF). For each pattern a set of weighted design pattern selection criteria has been defined. The user is responsible for identifying the components in the problem scenario and selecting suitable design pattern criteria and relationships for each identified component. Also user is asked to state the problem scenario and it is evaluated in Watson Assistant. Based on the selected criteria weightages and confidence received from the Watson assistant, appropriate design pattern is suggested with generated simplified class diagrams and the design reasoning. The tool will suggest only one design pattern. With the results of the survey conducted for novice developer, 84.8% of users were able to learn something related to design patterns by using the tool and for the test scenario tested the recommendations were 83.3% accurate. Further improvements can be suggested in the usability, accuracy, design reasoning and support, for more design patterns to reach the production level and additionally can also add more user interactions by introducing a virtual teacher as in the form of chat bot.

ACKNOWLEDGEMENTS

First and foremost, I would like to express my sincere gratitude to my research supervisor, Dr. Indika Perera for his continuous guidance, knowledge shared and patience extended towards my research.

And my sincere gratitude goes towards all my colleagues and friends for their kind support given to continue my research, by providing ideas and sharing their experiences.

Finally, I would also like to thank my parents for always encouraging and guiding myself towards wisdom with an unconditional love throughout my life.

TABLE OF CONTENT

DECLARATION	i
Abstract	ii
ACKNOWLEDGEMENTS	iii
TABLE OF CONTENT	iv
LIST OF FIGURES	vii
LIST OF ABBREVIATIONS	x
LIST OF APPENDICES	xi
INTRODUCTION	1
1.1 Design Patterns	3
1.2 Structure of the Design pattern.....	3
1.3 Design pattern selection criteria	4
1.4 Problem	4
1.5 Objectives	5
1.5.1 General Objectives.....	5
1.5.2 Specific Objectives	6
1.6 Suggested System.....	6
LITERATURE REVIEW.....	7
2.1 Introduction	8
2.2 Design patterns	8
2.2.1 Purposes and the Scopes of the design patterns.....	11
2.2.2 Relationship between Design patterns	12
2.2.3 Case studies on Design Patterns	20
2.3 Similar Researches	21
2.3.1 Automated Framework to Select Suitable Design Pattern.....	21
2.3.2 Recommendation System for Design Patterns.....	22
2.3.3 Design Pattern Recommendation.....	22
2.3.4 Interactive Design Pattern Recommendation	23
2.3.5 A solution strategy method for Design pattern selection.....	23

2.3.6	Design pattern selection based on MAS technology	24
2.3.7	Design Patterns Searching System using Case-based Reasoning.....	25
METHODOLOGY	26
3.1	Introduction	27
3.2	Proposed Solution.....	27
3.2.1	Graphical User Interface for user to interact with the system	27
3.2.2	Design pattern framework providing the best matching design pattern	27
3.2.3	Map user data with design pattern framework.....	28
SOLUTION ARCHITECTURE AND IMPLEMENTATION	29
4.1	Introduction	30
4.2	High level architecture	30
4.2.1	User Design Panel.....	31
4.2.2	Design pattern selection framework	32
4.2.3	Knowledge base	34
4.3	Implementation.....	40
4.3.1	Watson Assistant configuration	40
4.3.2	Graphical user interface-web page	43
4.3.3	Design pattern selection framework service	48
4.3.4	Knowledge base	49
DATA ANALYSIS AND EVALUATION	50
5.1	Introduction	51
5.2	Analysis on adding design patterns	51
5.3	Evaluation on design pattern selection criteria.....	53
5.4	Evaluation on developed sample prototype application.....	53
5.4.1	Evaluation of different problem scenarios.....	54
5.4.2	Evaluation on same scenario with different users	58
CONCLUSION	62
6.1	Research contribution.....	63
6.2	Research Limitation	64
6.3	Future Work and conclusion	64
References	66

APPENDIX A: Survey Questioner 01.....	70
APPENDIX B: Survey Questioner 02.....	77

LIST OF FIGURES

Figure 2.1: Design pattern Relationships.....	13
Figure 2.2: Categorization of design patterns relationships.....	14
Figure 2.3: Revised Classification.....	15
Figure 2.4: Layered structure of the design patterns.....	16
Figure 2.5: The structure of the Prototype-Abstract Factory.....	18
Figure 2.6: The frequency of occurrences of design pattern.....	19
Figure 2.7: Classification of Pattern Couplings.....	20
Figure 4.1: High level architectural diagram of the suggested system.....	30
Figure 4.2: High level overview of the User design panel.....	31
Figure 4.3: High level architecture of Design Pattern Framework.....	33
Figure 4.4: Added Intents in defined Watson Assistant skill.....	41
Figure 4.5: Adding examples to an intent.....	41
Figure 4.6: WA - Request JSON.....	42
Figure 4.7: WA Response JSON.....	42
Figure 4.8: Problem Description.....	43
Figure 4.9: Design Panel.....	44
Figure 4.10: Component creation model window.....	45
Figure 4.11: Design Pattern suggestion request.....	46
Figure 4.12: Generated UML diagram.....	47
Figure 4.13: Design reasoning.....	47
Figure 4.14: Design pattern suggestion request.....	49
Figure 4.15: ER diagram of the database.....	49
Figure 5.1: Participant count with respect to designation.....	51
Figure 5.2: Importance of adding design pattern to the program solution design....	52
Figure 5.3: Experience in applying design patterns.....	52
Figure 5.4: Pattern selection criteria usage.....	53
Figure 5.5: Watson assistant selected intents and confidence.....	56
Figure 5.6: Accuracy of the returned intent from Watson Assistant.....	56
Figure 5.7: Accuracy of the design pattern suggestion frame work.....	57

Figure 5.8 Completeness of the generated class diagrams.....	57
Figure 5.9: Usability of the developed application.....	58
Figure 5.10: User interaction involved in the form of thinking and analyzing the given scenario.....	59
Figure 5.11: Learning experience with identifying criteria to think when doing a solution design.....	59
Figure 5.12: Understandability of the generated pattern selection reasoning.....	60
5.13: Value addition to the knowledge.....	60
5.14: Developed application as a design pattern learning tool.....	61

LIST OF TABLES

Table 2.1: Design patterns its intent and aspects.....	8
Table 2.2: Purposes and the Scopes of the Design patterns.....	12
Table 2.3: Types of design patterns relationships and reasoning.....	14
Table 2.4: Relationships categorization.....	16
Table 4.1: Pattern categories and pattern criteria.....	35
Table 4.2: Pattern criteria and weightages.....	37
Table 5.1: Different problem scenario.....	54
Table 5.2: Evaluation Results on different problem scenarios.....	55

LIST OF ABBREVIATIONS

Abbreviation	Description
GoF	Gang Of Four
AST	Abstract Syntax Trees
DAO	Data Access Objects
DPR	Design Pattern Recommender
GQM	Goal-Question-Metric
XML	Extensible Markup Language
GSSMatrix	Global Semantic Similarity Matrix
QMP	Query-Matching-Pattern
QSPQ	Query-Similarity- Previous Query
QAS)	Question-Answer-Session
CIK	Collaborative-Implicit-knowledge
DPS	Design Pattern Selection
MAS	Multi-Agent System
FCA	Formal Concept Analysis
CBR	Case Based Reasoning
WA	Watson Assistant

LIST OF APPENDICES

Appendix	Description	Page
Appendix - A	Survey Questioner 01	70
Appendix - B	Survey Questioner 02	77

CHAPTER 1
INTRODUCTION

In today's context software systems are heavily used in all most all the domains. There are large numbers of software development companies all around the world to develop the software systems. Apart from the knowledge in business domain, it is important to have a well experienced development team in order to deliver a quality software product.

When it comes to software systems, complexity is increasing rapidly. Software systems become more complex while achieving extendibility, ease of maintenance and qualitative attributes such as performance, security, usability, accuracy. The best way to address this complexity problem would be to develop the system based on a quality software design. The software system design should be flexible, maintainable, extensible and reusable. This reveals the importance of a well-formed and accurate software system design.

The software system design is carried out by an experienced person, which is most of the time done by a software architect. But there is a responsibility for the developers to do code level designing. To engage in these tasks, it is required to have an understanding on coding standards, design patterns, architectural patterns, and system integration patterns and so on.

The first step in designing software would be, knowing the software design patterns and how to use them properly. This will reduce the development cost and maintenance cost. For a well experienced developer this will not be a big task but, for a new developer it would not be an easy task.

Selecting the most suitable design pattern is a critical issue that novice developers face when designing a solution for a problem. It is mainly due to largely available design patterns and lack of expertise knowledge required on design pattern selection.

1.1 Design Patterns

In the software engineering domain, there are number of design patterns which are available and those design patterns are used according to the problem domain. Design patterns are introduced to the code with the guidance of an expert of the software development practice.

Design patterns are a generic well proven solution to common design problems which are repeatedly used [1]. Expertise knowledge in designing area is a plus point where it leads to a product with high quality, which functions in an effective way. The benefits of the design patterns can be stated as increasing the flexibility, maintenance and reusability of software design, contributing to the extensibility of software, capturing the design knowledge based on hands on experience in software design, documenting the best practices which solves many different types of problems, a communication tool between software developers which provide a mechanism to share workable software solutions between developers and working organizations [2][3].

In general, a pattern has four essential elements [1] which are,

- The pattern name – which is used to describe a design problem
- The problem - which describes when to apply the pattern
- The solution – which describes the elements that used to make up the design, responsibilities, their relationships, and collaborations
- The consequences – which refers to results and trade-offs of applying the pattern

1.2 Structure of the Design pattern

Design patterns can be described using a consistent format. According to following template each pattern can be divided into sections. The template helps to form a

uniform structure to the information provided and compare it. This makes design patterns easier to learn, and use [1].

The template consists of the following sections.

- Pattern Name and Classification
- Intent
- Motivation
- Applicability
- Structure
- Participants
- Collaborations
- Consequences
- Implementation
- Sample Code
- Known Uses
- Related Patterns

1.3 Design pattern selection criteria

It might be hard to find the design pattern which addresses a design problem. To select the relevant design pattern, as the first step it is needed to consider how a design patterns can solve a design problem, for that it is required to go through the intent section which is mentioned under design pattern template. Then it is needed to figure out how the patterns are interrelated. Next is to figure out the purpose of the pattern which could be creational, structural or behavioral. Then it is needed to examine the design whether there is any reason for redesign. As the last step it is required to identify what could vary in the design that is made.

1.4 Problem

Once the problem domain is clearly defined the next step would be identifying major components of the system, relationships between components, and behaviors of the

components. Then the next step would be selecting and applying the most suitable design pattern or design patterns.

For an expert developer who has a keen knowledge of design patterns and its use cases, can easily select best fitting pattern to particular design problem [4]. But for a novice developer it is a different story. The main reasons are, novice developers do not have a clear understanding of the problem domain or tangible definition of it [5] and lack of knowledge in design patterns [4].

There should be a practical way of getting knowledge which facilitates novice developers on selecting most suitable design pattern for their software solution design.

1.5 Objectives

The prime objective of this research is to explore the existing researches carried out to recommend the most suitable design pattern to model a given software solution and to combine them with new findings related to selecting the appropriate design pattern and finally to come up with a tool which facilitates novice developers to model software solution with recommended design patterns.

1.5.1 General Objectives

- To come up with a design pattern recommendation tool which helps novice developers to model their software solution with recommended design patterns and get familiar with design patterns, selecting proper design patterns and its applicability.

1.5.2 Specific Objectives

- To come up with user interactive software solution designing panel which enables the user to,
 - Create objects/components
 - Assign values to the objects according to role of the objects/components and its behaviors
 - Make relationships between created objects/components
- To come up with design pattern understanding mechanism
- To come up with detailed analysis of the 23 design patterns [1].
- To come up with design pattern selection framework
- To come with an algorithm which matches design patterns and the software solution designed
- To come with an algorithm which priorities the design pattern recommendations made
- To come up with design pattern recommendation reasoning panel

1.6 Suggested System

To address the mentioned problem, the solution provided is a user interactive design pattern recommendation tool. The recommendation tool will provide graphical user interface to model the user problem in a graphical manner. This will be used to get the user data into the system.

A design pattern selection framework is introduced to the system to make pattern selection decisions. There is a mapping algorithm in order to map the user data with the design pattern selection framework.

CHAPTER 2
LITERATURE REVIEW

2.1 Introduction

The research works that have been done related to the problem domain were identified and has been thoroughly understood. In this section the research findings are arranged in an order for easy of understanding and for the clarity.

2.2 Design patterns

Even though there are plenty of design patterns available in the current context, this is only focused on the design patterns explained by GoF [1]. The below table is used to represent the design pattern intents and aspects as they have mentioned in the GoF [1]. The content of this table is directly taken from GoF [1] but under different topics.

Table 2.1 : Design patterns its intent and aspects.

Name	Intents	Aspect(s) That Can Vary
Abstract Factory	“Provide an interface for creating families of related or dependent objects without specifying their concrete classes.”	“families of product objects”
Adapter	“Convert the interface of a class into another interface clients expect. Adapter lets classes work together that couldn't otherwise because of incompatible interfaces.”	“interface to an object”
Bridge	“Decouple an abstraction from its implementation so that the two can vary independently.”	“implementation of an object”
Builder	“Separate the construction of a complex object from its representation so that the same construction process can create different representations.”	“how a composite object gets created”

Chain of Responsibility	“Avoid coupling the sender of a request to its receiver by giving more than one object a chance to handle the request. Chain the receiving objects and pass the request along the chain until an object handles it.”	“object that can fulfill a request”
Command	“Encapsulate a request as an object, thereby letting you parameterize clients with different requests, queue or log requests, and support undoable operations.”	“when and how a request is fulfilled”
Composite	“Compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly.”	“structure and composition of an object”
Decorator	“Attach additional responsibilities to an object dynamically. Decorators provide a flexible alternative to subclassing for extending functionality.”	“responsibilities of an object without subclassing”
Facade	“Provide a unified interface to a set of interfaces in a subsystem. Facade defines a higher-level interface that makes the subsystem easier to use.”	“interface to a subsystem”
Factory Method	“Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses. ”	“subclass of object that is instantiated”
Flyweight	“Use sharing to support large numbers of fine-grained objects	“storage costs of objects”

	efficiently.”	
Interpreter	“Given a language, define a representation for its grammar along with an interpreter that uses the representation to interpret sentences in the language.”	“grammar and interpretation of a language”
Iterator	“Provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation.”	“how an aggregate's elements are accessed, traversed”
Mediator	“Define an object that encapsulates how a set of objects interact. Mediator promotes loose coupling by keeping objects from referring to each other explicitly, and it lets you vary their interaction independently.”	“how and which objects interact with each other”
Memento	“Without violating encapsulation, capture and externalize an object's internal state so that the object can be restored to this state later.”	“what private information is stored outside an object, and when”
Observer	“Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.”	“number of objects that depend on another object; how the dependent objects stay up to date”
Prototype	“Specify the kinds of objects to create using a prototypical instance, and create new objects by copying this prototype.”	“class of object that is instantiated”
Proxy	“Provide a surrogate or placeholder for another object to control access to	“how an object is accessed; its location”

	it.”	
Singleton	“Ensure a class only has one instance, and provide a global point of access to it.”	“the sole instance of a class”
State	“Allow an object to alter its behavior when it’s internal state changes. The object will appear to change its class.”	“states of an object”
Strategy	“Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it.”	“an algorithm”
Template Method	“Define the skeleton of an algorithm in an operation, deferring some steps to subclasses. Template Method lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure.”	“steps of an algorithm”
Visitor	“Represent an operation to be performed on the elements of an object structure. Visitor lets you define a new operation without changing the classes of the elements on which it operates.”	“operations that can be applied to object(s) without changing their class(es)”

2.2.1 Purposes and the Scopes of the design patterns

Design patterns have a scope and the scope can be either class level or object level. This is the applicability of the design patterns. Based on the purpose the design patterns have been categorized as Creational, structural and Behavioral.

Table 2.2 : Purposes and the Scopes of the Design patterns

		Purpose		
		Creational	Structural	Behavioral
Scope	Class	Factory Method	Adapter (class)	Template Method Interpreter
	Object	Abstract Factory Builder Prototype Singleton	Adapter (object) Bridge Composite Decorator Flyweight Facade Proxy	Chain of Responsibility Command Iterator Memento Mediator Observer Strategy State Visitor

2.2.2 Relationship between Design patterns

2.2.2.1 Relationships between Design Patterns

This section summarizes the research done by Zimmer [6]. They have discussed about organizing design pattern relationships in to different categories, revise the design patterns and their relationships and arranging them in to different layers.

They have represented a catalog for design patterns, and it was done according to the following criteria.

1. Jurisdiction (class, object, compound)
2. Characterization (creational, structural, behavioral)

Their major research outcomes were as follows:

- Done a classification for relationships between design patterns
- A new generalized design pattern formed using several other design patterns

Table 2.3: Types of design patterns relationships and reasoning

Relationship	Reason
X uses Y in its solution	The problem addressed by X has a sub problem which is similar to the problem addressed by Y Solution Y is a part of solution X
X is similar to Y	X and Y addressing similar kind of problems but not the similar solutions.
X can be combined with Y	The most commonly the combination of X and Y

After doing the classification, the categorization of relationships is visualized as in figure 2.2.

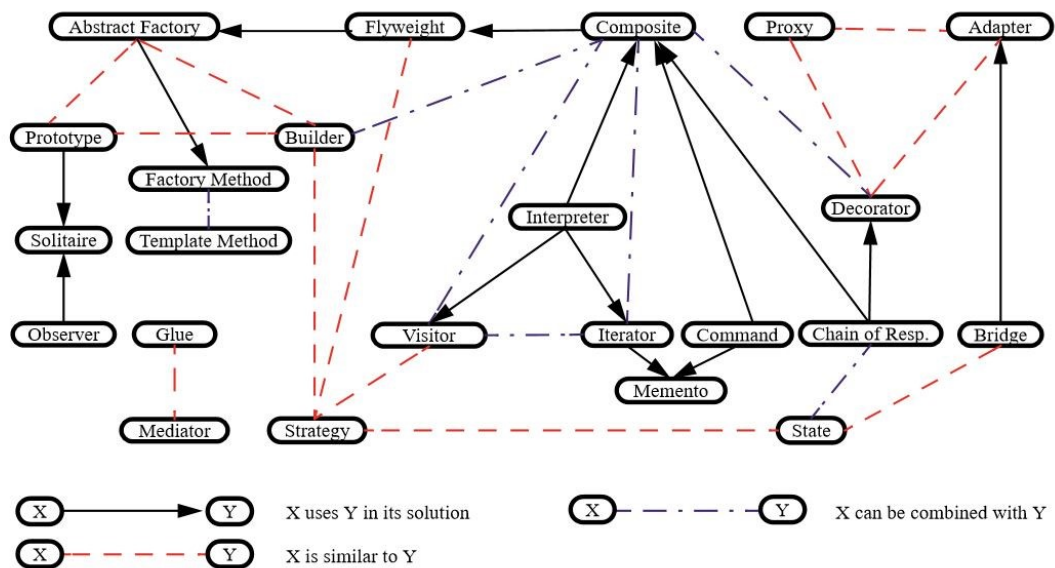


Figure 2.2: Categorization of design patterns relationships

Source: [6]

They have come up with a new design pattern called, Objectifier which is responsible for objectification of behaviors in design patterns.

Two design patterns can be related in different ways, a pair of patterns like Abstract Factory / Prototype can be combined as similar patterns. So, it is difficult to organize the relationships in different categories. In order to assign relationships to the most adequate category, the classification of relationships is revised.

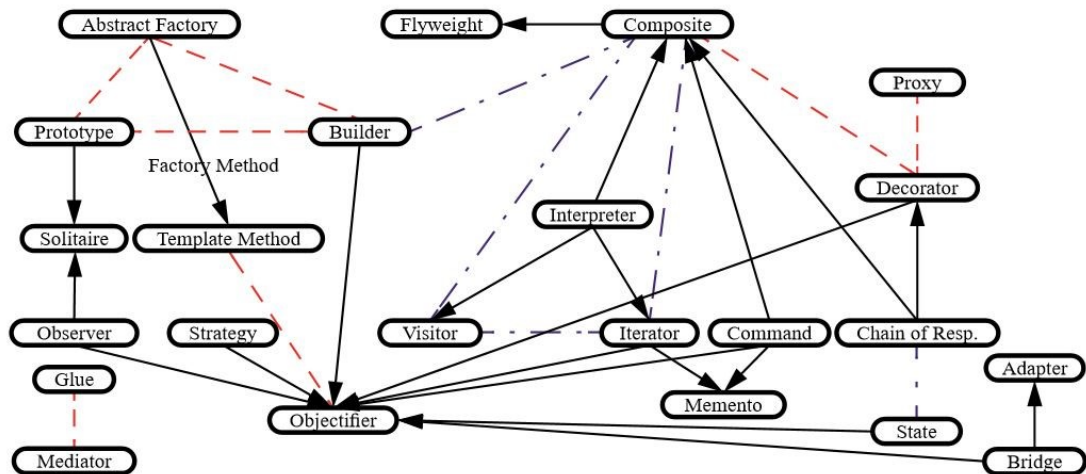


Figure 2.3: Revised Classification

Source: [6]

According to the Figure 2.3, it can be shown that the “X uses Y” has become the most frequent relationship. Based on this relationship the design patterns were divided in to three different layers, which are called:

- Basic design patterns and techniques.
- Design patterns which address typical software problems.
- Design patterns which specific to an application domain.

The layered structure of the design patterns is represented in figure 2.4.

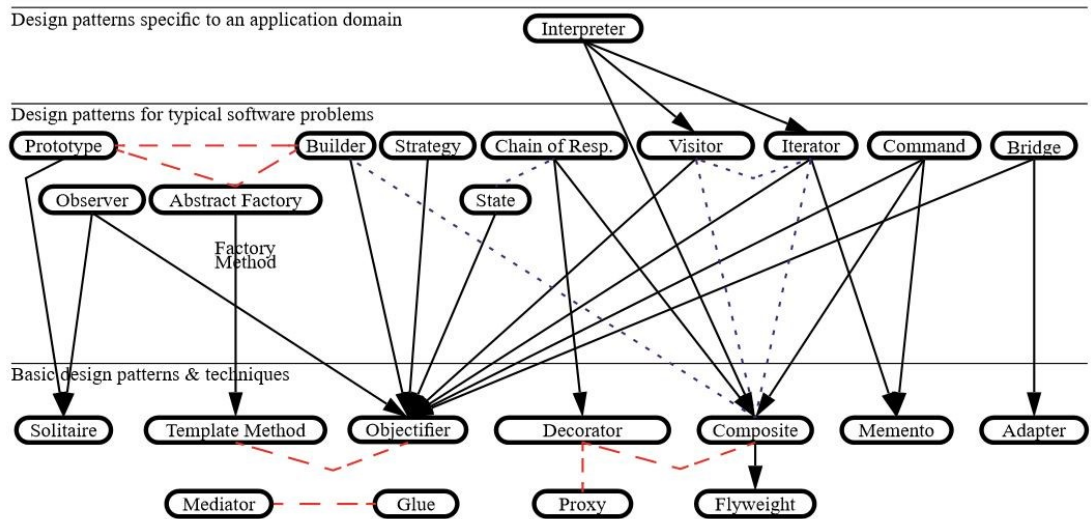


Figure 2.4: Layered structure of the design patterns

Source: [6]

2.2.2.2 Relationships classification between Object-Oriented Design Patterns

Summarization of the research done by Noble [7] is as follows. They have proposed a two-level classification scheme containing Primary Relationships and Secondary Relationships. Primary relationships were categorized in to three sections as uses, refines, and conflicts, whereas secondary relationships were categorized in to nine sections.

Table 2.4: Relationships categorization

Source: [7]

Primary Relationships	
Uses	“One pattern uses another pattern”
Refines	“A specific pattern refines a general pattern”
Conflicts	“A pattern addresses the same problem as another pattern”
Secondary Relationships	
Used by	“A smaller pattern is used by a larger pattern”
Refined by	“A general pattern is refined by a specific pattern”
Variant	“A variant pattern refines a more well-known pattern”
Variant Uses	“A variant of one pattern uses another pattern”

Similar	“A pattern is similar to another pattern”
Combines	“Two patterns combine to solve a single problem”
Requires	“A pattern requires the solution of another pattern”
Tiling	“A pattern uses”
Sequence of Elaboration	“A sequence of patterns from the simple to the complex”

2.2.2.3 Composite Design Patterns

The following sections are summarization of the research done by John [8]. The report contains details regarding the composite design patterns and a summary of this report is discussed below.

Template Method-Factory Method

The template method separates an operation in to two parts, variant and invariants. The variant parts are also called as primitive operations, which are defined in a subclass. It specifically considers on responsibility. But factory method defers behaviors to a subclass.

Both patterns have the scope as class; therefore these patterns are less flexible but tend to be simpler and more efficient. These results in a light weight application at both compile and run time.

Prototype-Abstract Factory

Instead of sub classing Abstract Factory to parameterize types of the product, Prototype can be used. In order to do a single concrete a Factory is needed to configure the appropriate prototype. Even though it is costly, using prototype can reduces the number of classes that are introduced by the Abstract factory.

“As a rule, PROTOTYPE will work wherever FACTORY METHOD will, and with more flexibility, but also with higher run-time cost.”

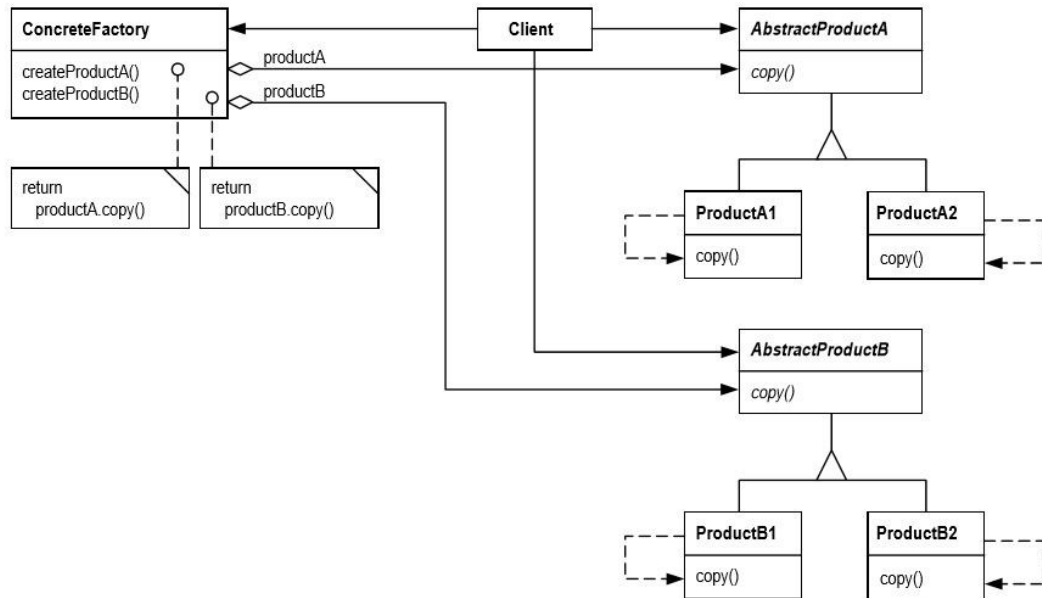


Figure 2.5: The structure of the Prototype-Abstract Factory

Source : [8]

Composite-Decorator

Composite implements the Component interface and it defines what you can treat uniformly. For Decorator to work properly it is needed to have a common interface. It is hard to decorate a component transparently unless both component and decorator share an interface.

Composite-Decorator composition adds weight to the argument for a uniform interface.

Composite-Flyweight

The Composite pattern can produce a lot of overhead if it is applied at too fine in granularity. Because it will create components for each element even though, those elements will be redundant. So, in such situations Flyweight can be applied because it reduces the redundancy by avoiding component creation for duplicated elements. But sharing components can cause problems.

Composite-Iterator-Visitor

Iterator traverse through the composites without thinking how they are linked together and it enables reuse of common traversal. Visitor lets you perform (or not perform) type specific work at each point in the traversal. But it seems like classes which are closely coupled would need someone to be responsible on coupling.

2.2.2.4 Coupling of Design Patterns: Common Practices and Their Benefits

This is the summarization of the research work done by William, James and Bieman [9]. In this paper they have done a qualitative assess on goodness of design pattern coupling by considering effects on maintainability, reusability and factorability. 16 papers were used to analyze the 23 design patterns in the GoF[1]. The analysis was based on the pattern occurrences and pattern coupling. Pattern coupling was categorized under coupling types such as tightly and loosely. The interaction types were coupled as intersection as “talks to” or “uses a”, composite and embedded as “has a”.

Pattern	Pattern Group		
	Creational	Structural	Behavioral
Factory Method	6		
Abstract Factory	7		
Builder	3		
Prototype	1		
Singleton	6		
Adapter		4	
Bridge		2	
Composite		8	
Decorator		2	
Facade		3	
Flyweight		2	
Proxy		5	
Chain of Responsibility			2
Command			4
Interpreter			2
Iterator			3
Mediator			5
Memento			1
Observer			11
State			4
Strategy			8
Template			6
Visitor			4
Totals	23	26	50

Figure 2.6: The frequency of occurrences of design pattern

Source: [9]

Coupled Patterns	Subtype			Strength	
	Intersection	Composite	Embedded (parent listed 1st)	Loose	Tight
Strategy-Abstract Factory	1			1	
Visitor-Template	1			1	
Builder-Template-Strategy	1			1	
Abstract Factory-Factory	1			1	
Mediator-Observer	1			1	
Composite-Visitor	1			1	
Factory-Prototype	1			1	
Singleton-State-Command	2				2
Singleton-State-Observer	1				1
Composite-Decorator-Flyweight	1				1
Observer-Facade-Template	1			1	
Bridge-Proxy-Flyweight	1			1	
Facade-Observer	1			1	
Mediator-Strategy	1				1
Adapter-Strategy	1				1
State-Singleton	1				1
Interpreter-Builder-Abstract Factory		1			1
Visitor-Command		1			1
Composite-Proxy		1			1
Observer-Singleton		1			1
Strategy-Observer-Composite			1	1	
Bridge-Observer-Proxy-Abstract Factory-Factory			1	1	
Mediator-Observer-Chain of Responsibility-Composite			1		1
Composite-Interpreter			1		1
Totals	17	4	4	12	13

Figure 2.7: Classification of Pattern Couplings

Source: [9]

2.2.3 Case studies on Design Patterns

2.2.3.1 Visitor versus Interpreter Pattern

The following is the summarization of the research done by Mark, Paul, Tijs and Jurgen [10]. In this paper they have discussed about selection of Visitor and Interpreter design patterns which are based on Abstract Syntax Trees (AST). AST is the based interpreter for Rascal programming language. They have used quantitative methods to understand the consequences of selection choices made.

They have gone through five realistic scenarios related with maintainers and have come up to a conclusion. The solution implemented with Visitor pattern is more maintainable than a solution implemented using Interpreter Pattern. But in some trivial situations, solutions based on the Interpreter pattern are more maintainable.

Further, they have mentioned the importance of giving attention towards the consequences when selecting design patterns.

2.2.3.2 Strategy Design Pattern

In this paper the author describes strategy pattern and its usage in .net frame work and according to them strategy pattern defines a family of algorithms that encapsulates one another, making sure that the algorithms are interchangeable within the family it has been developed into[11]. It also enables an algorithms behavior to be selected at runtime [11].

2.2.3.3 Strategy Pattern as Payment Pattern for Internet Banking

They have analyzed the internet banking system and how the payment process is done [12]. With the support of analyzed details they were able to formalize a payment pattern which is equal to Strategy pattern [12].

2.2.3.4 Factory Design Pattern for Database Connection and Daos in Struts Framework

This paper discusses the way that the factory design pattern can be used to manage data objects and database connection in Struts Framework [13]. The reason behind selecting Factory pattern was its capability of separating application and a family of classes [13]. It is a simple way where the family of products can be extended with minor changes in application code [13].

2.3 Similar Researches

2.3.1 Automated Framework to Select Suitable Design Pattern

This section summarized research paper done by Rizwan and Waleed [14]. They have proposed an automated framework to select suitable design pattern based on the attributes of the design patterns which are mentioned by Gang of Four authors (GoF) . They use two repositories called New Repository and original repository. New Repository contains problems and most relevant design pattern solutions for those problems. Original repository contains the attributes of the design patterns which helped to identify the better solution for a given problem. The attributes are pattern name, structure, intents, applicable and descriptions.

First a developer needs to express his problem as a question and checks against the new repository. If similar matching problem is found, then an output will be provided with the relevant design pattern solution. If no match is found within the new repository then the problem will be matched with the original repository (GoF repository). To select criteria from the Original repository it is needed to answer some questions and those questions are built upon the properties of the design patterns.

2.3.2 Recommendation System for Design Patterns

Following is the summarization of the research done by Palma, Farzin, Guéhéneuc and Moha [15]. Design Pattern Recommender (DPR) prototype is proposed to suggest design patterns and it is based on the Goal-Question-Metric (GQM) approach. Maximum number of questions that the system may ask is 11. DPR uses a ranking based selection approach. It is a two-fold solution. In the primary-level, DPR proposes design patterns for a specific problem context. In the secondary level, based on the recommendations one or more design patterns will be implemented for the initial model by the developer. They also have a knowledge base used for the DPR process. This was inspired from previous work.

2.3.3 Design Pattern Recommendation

This section summarizes the research done by Suresh, Naidu and Kiran [16]; They proposed a system to evaluate the user scenarios effectively with the help of information retrieval techniques and recommends pattern. As the recommendation technique, Social recommendation is used. It recommends patterns based on the past behavior of similar users. Two search scenarios are provided to find the suitable pattern. In scenario 1, find the intent of the user query and retrieve appropriate questions related to the intent from the pattern repository. While giving the answers, points are given and based on the points a recommendation is given. In scenario 2, search for the similar type of query from the history database. If a match is found

then suggest the pattern, if not continue with the scenario 1. The used pattern repository is based on the XML.

2.3.4 Interactive Design Pattern Recommendation

Summarization of the research done by Issaoui, Bouassida, and Abdallah [17] as follows. They have proposed user interactive, semi-automated solutions to suggest design patterns which better fits with the designer's design. Design structure and intention were their major concerns. It is a two -phase approach. As the first approach semantic similarity matrix is calculated and it is called as GSSMatrix (Global Semantic Similarity Matrix). GSSMatrix is used to approximate the similarity between the developer drawn design and design patterns available in its knowledge base. In GSSMatrix keywords of the patterns are represented by the lines and terms of patterns are represented by the columns. GSSMatrix scores are calculated for all available design patterns and max score is used to determine the best fitting design pattern. The second phase is user interactive, and the system is interacted with the developer by asking questions. The questions were asked with the intention of reformulating the intentions of the design patterns which were selected in the first phase. Based on developer's answer the recommendation system lists the possible design patterns for recommendation. This process is repeated until best matching design pattern is selected.

2.3.5 A solution strategy method for Design pattern selection

The brief summary of the research done by Sahly and Sallabi [18] is given under this section. In the proposed approach they have categorized the system users into three levels which are Novice, advanced beginner and Expert. This categorization is based on answers of three questions. Once the user level is identified pattern recommendation is done. It is based on four iterative steps which are Identify Design Problem, Retrieving Patterns, Recommendations and Evaluations.

In the first step, identify the design problem through queries submitted. To retrieve patterns four algorithms are used and those are Query-Matching-Pattern (QMP) to find the similarity between intents and the given query, Query-Similarity. Previous Query (QSPQ) to find the similarities between requested query and previous users' query. Question-Answer-Session (QAS) to narrow down the selection process by finding patterns based on question-answering and Collaborative-Implicit-knowledge (CIK) to transfer of implicit knowledge through collaboration with communities of users to find the pattern. In the third step it provides three types of recommendations. Those are the recommending patterns, recommending how to apply patterns and Recommending pattern sequences. As the last step evaluation is done through user feedback on the system in the form of question-answering.

2.3.6 Design pattern selection based on MAS technology

The summary of the research done by Eiman, Salah, Maha, Zabata and Omar [19] is given in the following paragraphs. The paper describes new design pattern architecture (DPS) to select design pattern and it is based on a Multi-Agent System (MAS). Their goal is to provide most relevant design pattern recommendation in order to reduce development efforts, assist and facilitate the new developers in selecting the most appropriate design pattern for their problem. This is a continuous work on the research proposed in [10]. MAS architectures can be represented as a social organization of independent software. This independent software's are called as an agent. They are flexible, reasonable, autonomous, learnable and cooperate with the environment. There are three layers which are Interface Layer to establish the interaction between user and the system, Application Services Layer to hold eight components which are used to implement the core functionality of the system and Infrastructure Layer to provide access to the data. They have used ten agents and each agent has their own responsibility of performing specific functions.

2.3.7 Design Patterns Searching System using Case-based Reasoning

The summarization of the research conducted by Muangon and Intakosum [20] is in following paragraphs. In this approach they have used both Formal Concept Analysis (FCA) and Case Based Reasoning (CBR). CBR is a powerful tool which is used in problem solving systems and FCA is a technique used to analyze data. These two approaches have been used to solve the index limitation Problem. When the system gets a new user problem description, then the system retrieves similar cases which are support for the problem. For that similarity functions are used. The output solutions are suggested to the user and those outputs are stored in the system as new knowledge which can be used to solve new problems. If the solutions suggested are not satisfying the user expectations, then system provides alternative methods by using FCA. As the first step FCA discovers related cases and the description of those related cases are presented to the user. In the second step relevant indexes are generated in order to complete the problem description. As the final step CBR is used to keep new experiences in data retrieval and revising processes which enables solving similar kind of problem in the future. Evaluation was done using prototype technique.

CHAPTER 3
METHODOLOGY

3.1 Introduction

In fulfilling the research outcome, the work has been done under three major sections, which are getting required data from the user in an interactive manner, defining a framework to select design pattern and lastly developing an algorithm to map user data with design pattern framework. This would suggest the best suited design patterns to the user.

3.2 Proposed Solution

For the proposed solution, a user interactive design pattern recommendation tool was built which consisted of three major sections. Those are;

- Graphical User Interface for user to interact with the system
- Design pattern framework providing the best matching design pattern
- Map user data with design pattern framework

3.2.1 Graphical User Interface for user to interact with the system

A graphical user interface is provided to the user in order to interact with the system. It has a design panel which enables the user to design his software solution with objects and their relationships.

Each component needs to select the relationships with other components and the pattern criteria which match with the component. The required user inputs are defined in a way in which it helps to map user data with the design pattern selection framework.

3.2.2 Design pattern framework providing the best matching design pattern

In order to build up a design pattern selection framework it is required to have a deep analysis on previous research done as well as the own findings. According to the research papers that have been followed, there are few concerns that we need to focus on. The concerns are the relationships between the design patterns, combinations of design patterns, tradeoff between combining design patterns and the consequences of using them. Therefore, the design pattern selection framework is built based on two major concerns, those are;

- Relationships between design patterns
- Consequences of applying design patterns

3.2.3 Map user data with design pattern framework

There are algorithms to map user data in to design pattern selection framework. The problem descriptions entered by the user is checked against the defined Watson assistant intent. If it is matched, the entity and the confidence recoded will be considered for further use. The user selected pattern criteria are used to identify the possible patterns. Using the Watson Assistant response and the selected pattern criteria the most suitable pattern is selected.

User input given under the relationships and the common criteria are used to determine the object, which is a possible candidate for the object in the suggested design pattern's class diagram. If any matches are found it will replace the default object in the class diagram and if not, it will show the default class diagram for the selected pattern.

Once the pattern is selected, the design pattern selection reasoning will be provided to the user.

CHAPTER 04

SOLUTION ARCHITECTURE AND IMPLEMENTATION

4.1 Introduction

In implementing the suggested solution, four main sections have been identified,

1. Graphical User Interface
2. Watson Assistant
3. Pattern Selection Algorithm
4. Pattern Selection Knowledge Base

4.2 High level architecture

The below diagram depicts the suggested architecture diagram of the system.

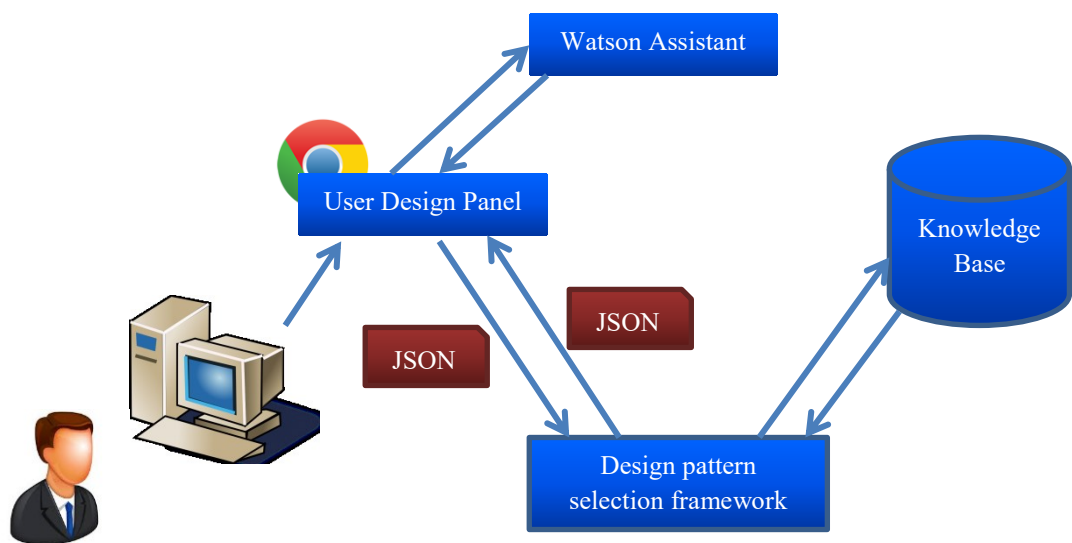


Figure 4.1: High level architectural diagram of the suggested system

The user can provide his problem in the User design panel by creating objects as components and mentioning their interconnections. In this process the system gives options to the user to select an order to specify the object description. Data is communicated in the form of JSON and the data is passed to the Design Pattern selection framework. Based on the user's inputs the system identifies the object and using the knowledge base and the defined algorithms the best matching design pattern is suggested. The UML class diagrams of the suggested pattern and the reasoning are composed in to a response and finally the response is passed back to the GUI.

4.2.1 User Design Panel

The user design panel is mainly compact with four sections,

1. Problem description
2. Design panel
3. Suggested design pattern with UML class diagram
4. Reasoning description

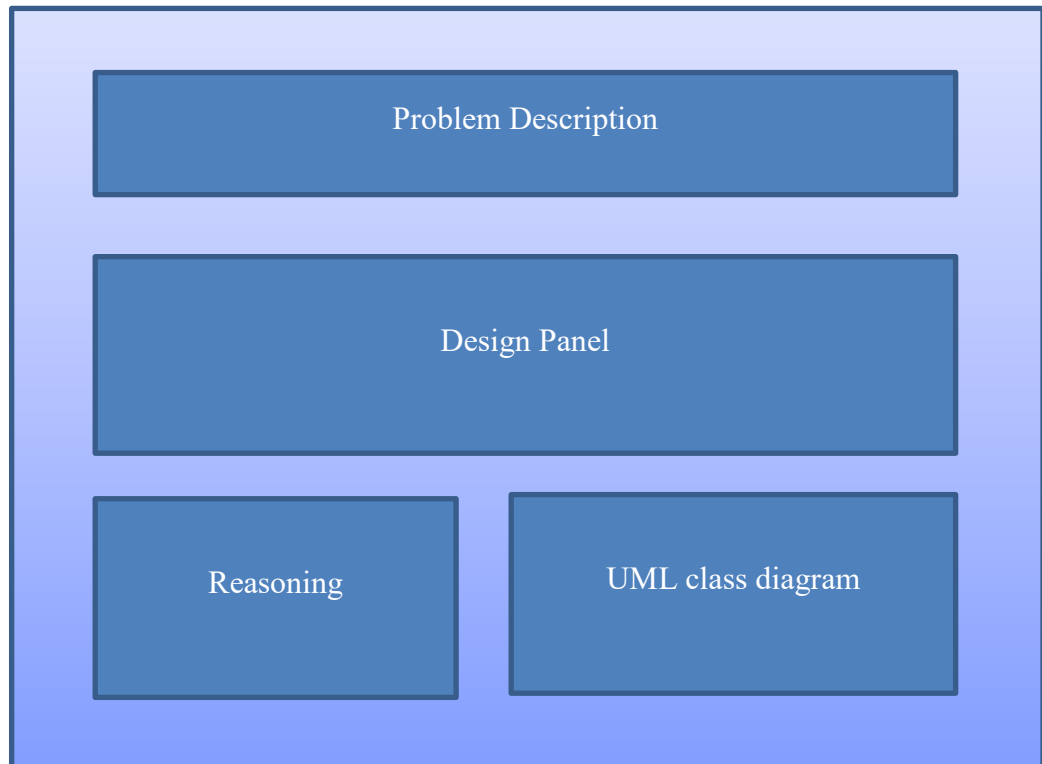


Figure 4.2: High level overview of the User design panel

4.2.1.1 Problem Description

The user is required to enter the problem description which is going to be model in the design panel. Based on the problem description provided, candidate intent is displayed. If it aligns with the problem description the user can select the intent.

4.2.1.2 Design Panel

Design panel is responsible for getting user inputs and making the request with all the components. Their relationships are sent to the Design pattern selection framework.

In order to proceed with designing the system, the user should understand the problem context and should analyze each component and its relationships. Thereafter the system can help the user to model the problem in the design panel.

Using the user interface, a user can create components in the design panel. When a component is created the system itself prompt options to the user where he can add definition to the component created. The component definition includes details about component creation, structuring, behaviors and relationships.

4.2.1.3 UML class diagram

The suggested pattern would be displayed in the form of UML class diagram for the user to have more clarity on the design pattern structure.

4.2.1.4 Reasoning

Simple descriptive reasoning is included to understand why the pattern is considered as the best matching pattern.

4.2.2 Design pattern selection framework

Design pattern selection framework is built based on the data in the knowledge base. The pattern selection algorithm is based on the problem description, pattern criteria rank, total pattern rank score, related patterns and the consequences of using them. The frame work can be divided in to following subsections,

1. Request manipulator
2. Design pattern selector
3. UML class generator
4. Response generator

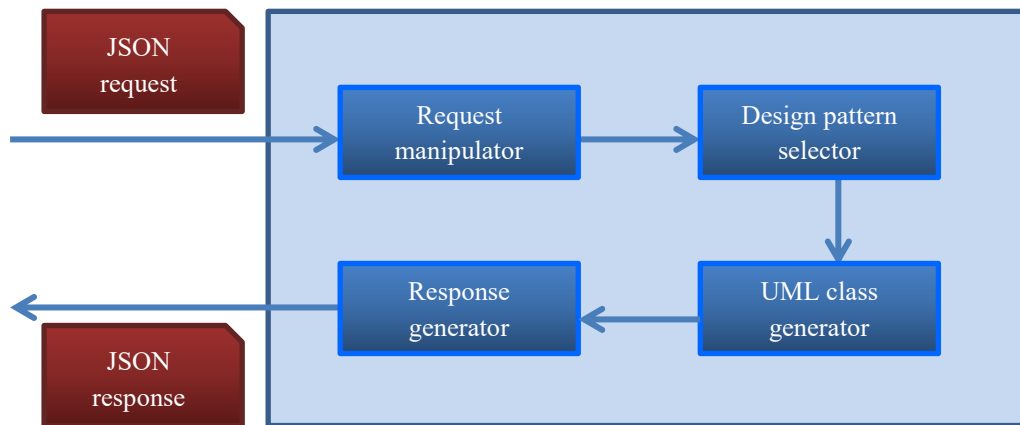


Figure 4.3: High level architecture of Design Pattern Framework

4.2.2.1 Request manipulator.

The attributes and its values in the JSON request are mapped to request manipulation object. The object is used for further processing and more values are added to it during the processing operations that are used to create the response JSON as well.

4.2.2.2 Design Pattern Selector.

Design pattern suggestion is made by going through three steps,

1. Design pattern suggestion based on the problem description
2. Design pattern suggestion based on the design pattern criteria
3. Design pattern suggestion based on the related pattern

4.2.2.2.1 Design pattern suggestion based on the problem description

Problem description is matched with candidate design pattern intents if any. This will help to suggest design pattern for the problem directly. The candidate design patterns are recorded in the request manipulation object to further use.

4.2.2.2.2 Design pattern suggestion based on the design pattern criteria

For each design pattern there is a set of criteria which is used to distinguish a pattern. These criteria are named as design pattern criteria. Each design pattern criteria are assigned with a rank. The same pattern criteria can be used to define more than one pattern and can have different ranking as well.

The possible design pattern per component is selected based on the pattern criteria selected per component by the user. If the same design pattern is suggested for more than one component, then those components are considered as one set of connected components which matched with the same suggested pattern. The possible design patterns and their scores are calculated by considering the pattern ranks which are recoded in the request manipulation object.

4.2.2.2.3 Design pattern suggestion based on the related pattern

If there is more than one suggestion for a component/set of components it checks the previously suggested design pattern for the same scenario and check for the related patterns. If a related pattern is found, then the related pattern is accepted as the best matching design pattern.

4.2.2.3 UML class diagram generator

Using the values set in the request manipulation object, the UML class diagram is generated as a JavaScript file. The generated UML diagram has the components model by the user as the objects. Based on the given information descriptions the UML diagram is decided. UML diagram information is stored in the request manipulation object.

4.2.2.4 Response generator

Suggested design pattern/s and the UML class diagram information are taken from the request manipulation object and those are added to the response object. Apart from that, reasoning description is generated using both request manipulation object details and the relevant reasoning information in the database. The response object is sent back to the GUI in the form of JSON.

4.2.3 Knowledge base

The knowledge base focuses on two major areas as, how user understand the scenario and how to select the best matching design pattern for the scenario.

Watson assistant is used to identify the intention of the user input. In this case the user input is the problem description as of the user's perspective which is the way that the user has understood the problem. To capture the varying problem

descriptions, even for the same scenario, a skill set was added to WA and it was trained with possible examples of expressing the same idea in different ways. The skill set can be exported as a JSON file.

As the second area, identification of the best matching design pattern by defining a set of criteria has been done. For each design pattern possible criteria have been defined and weighted accordingly.

The process of adding weightages was done through set of iterations of revisiting all [criteria and testing with scenarios where the pattern needs to be applied was known.

The criteria defined were organized under two levels, which mean the criteria were put under different categories. The categories, design pattern criteria [21]-[30] and their weightages are depicted in the below tables.

The weightages are assigned based on the relevance to the design pattern. Weightage of 8 is given when the specific criteria is enough to suggest the design pattern. If a criterion is somewhat relevant but not specific to the design patterns, those are weighted with 1. The maximum weightage 8 is determined by assuming a pattern that would have less than 8 nonspecific but relevant pattern criteria. If it is more than 8, then the maximum weightage value must be reassigned to a higher value.

Table 4.1: Pattern categories and pattern criteria

Pattern Category	Pattern criteria
Object structure	Less number(less than 5) of constructor arguments Has incompatible Interface. Resolve incompatible interface. Required object aggregation to have the complete object.
Object creation	Consider the new operator as harmful Subclass selection happens at runtime.(polymorphism) Object creation happens, step by steps.

	Object creation can happen on demand/may not need all the time.
Object count	Allow to create multiple instances. Required only one instance with global point of access to it. The number of objects required is high.
Object representation	Have multiple representations. Large numbers of subclasses are possible, mostly created to represent different appearance. The appearance can be added as a responsibility dynamically.
System count	System consists of subsystems and belongs to a subsystem. System consists of subsystems and connected with subsystem classes.
Work	Do complex communication between connected objects. Have so many relationships/connections. Access rights are granted based on the user. Do update observers. Get updates on specific values.
Object creation cost	Object creation is expensive. Use IO/DB connections.
Object state	Need to store previous state of the object. Behavior is based on a state.
Consequences	Application doesn't need object identity. Some features of the object can be reused.
Algorithm	Use algorithms to select at run time Have identical classes which work on variation of the same algorithm. Same algorithm steps can be sub classed
Implementation	The object implementation should be selected at run time
Collection	Many unrelated operations are required. Add operations frequently.

Table 4.2: Pattern criteria and weightages

Pattern criteria	Singleton	Factory Method	Builder	Abstract Factory	Prototype	Adapter	Bridge	Composite	Decorator	Facade	Proxy	Flyweight	Chain of responsibility	Command	Interpreter	Iterator	Mediator	Memento	Observer	State	Strategy	Template Method	Visitor
Need only one instance with global access to it.	8	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Has a super class	0	8	0	1	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Object creation selected at run time	0	8	0	1	1	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0
Object is Complex	0	0	8	0	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Object is created part by part	0	0	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Object creation involves with algorithms/preprocessing etc	0	0	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Has multiple representations	0	1	8	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Has handful(5) of constructor parameters	0	0	8	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
Has families of related/dependent objects	0	0	0	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Has parallel class Hierarchies	0	0	0	8	0	0	0	0	0	0	0	0	1	0	0	0	1	0	1	0	0	0	0
Object creation is expensive	0	0	0	0	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Use Io/Db connections	0	0	0	0	8	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1

Has incompatible Interface	0	0	0	0	0	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Resolve incompatible interface	0	0	0	0	0	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
The object implementation should be selected at run time	0	0	0	0	0	0	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Required object aggregation to have the complete object.	0	0	0	0	0	0	0	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Large number of subclasses are possible, mostly created to represent different appearance	0	0	0	0	0	0	0	0	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0
The appearance can be added as a responsibility dynamically.	0	0	0	0	0	0	0	0	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0
System consists of subsystems and belong to a subsystem	0	0	0	0	0	0	0	0	0	8	0	0	0	0	0	0	0	0	0	0	0	0	0
System consists of subsystems and connected with subsystem classes	0	0	0	0	0	0	0	0	0	8	0	0	0	0	0	0	0	0	0	0	0	0	0
Object creation can happen on demand/may not need all the time.	0	0	0	0	0	0	0	0	0	0	8	0	0	0	0	0	0	0	0	0	0	0	0
Access rights are granted based on the user	0	0	0	0	0	0	0	0	0	0	8	0	0	0	0	0	0	0	0	0	0	0	1
The number of objects required is high(more than 100).	0	0	0	0	0	0	0	0	0	0	0	8	0	0	0	0	0	0	0	0	0	0	0
Application doesn't need object identity.	0	0	0	0	0	0	0	0	0	0	0	8	0	0	0	0	0	0	0	0	0	0	0
Some features of the object can be reused.	0	0	0	0	0	0	0	0	0	0	0	8	1	0	0	1	0	0	0	0	0	0	0

configured by the chat bot application developer. In this prototype the concept of identifying intent is used in identifying the intent of the problem provided by the user under problem description.

In WA, design patterns are configured as intents under a skill set and possible word phrases are given for the training.

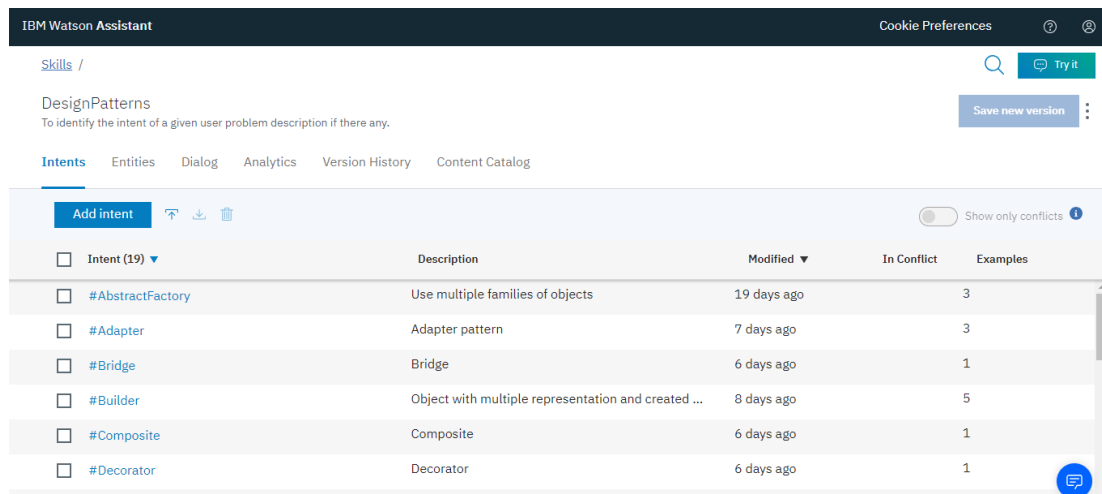


Figure 4.4 : Added Intents in defined Watson Assistant skill

As an example, consider the builder pattern. To check if any user input has the intent similar to the builder pattern, possible ways of expressing the same intent should be added as example for the builder entity.

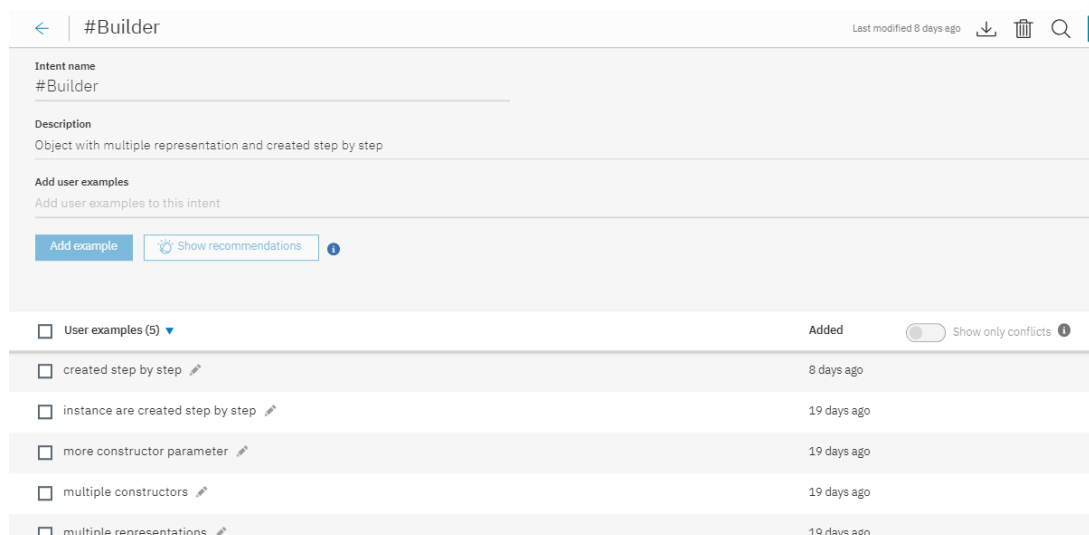


Figure 4.5 : Adding examples to an intent

The WA can be accessed via REST api calls. The supported request/response format is JSON.

As an example, consider a scenario where we need to convert a PDF document to a Word document.

The problem description would be similar to this;

“Given PDF document should be converted to a Word document. The PDF document is comprised of complex objects which have multiple representations as well. Text, images, tables should be identified correctly and should be convert to word supported format”

The request JSON is shown in figure 4.6.

```
{
  "input" : {
    "text" : "Given PDF document should be converted to a Word document. The PDF document is comprised of complex objects which have multiple representations as well. Text, images, tables should be identified correctly and should be convert to word supported format"
  },
  "context" : {
  }
}
```

Figure 4.6: WA - Request JSON

The response JSON is shown in figure 4.7.

```
{
  "intents": [
    {
      "intent": "Builder",
      "confidence": 0.6227362155914307
    }
  ],
  "entities": [],
  "input": {
    "text": "Given PDF document should be converted to a Word document. The PDF document is comprised of complex objects which have multiple representations as well. Text, images, tables should be identified correctly and should be convert to word supported format"
  },
  "output": {
    "generic": [
      {
        "response_type": "text",
        "text": "Builder pattern is selected"
      }
    ],
    "text": [
      "Builder pattern is selected"
    ],
    "nodes_visited": [
      "node_2_1552590226984"
    ],
    "log_messages": []
  },
  "context": {}
}
```

Figure 4.7: WA Response JSON

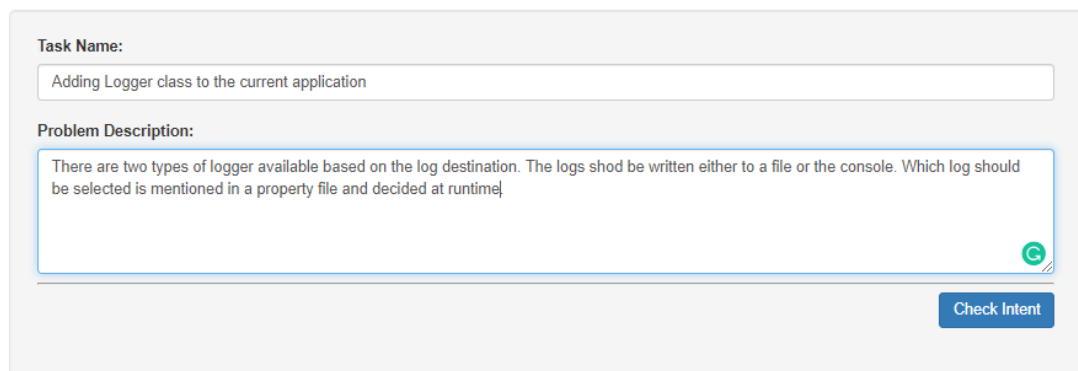
The important information extracted from the response is the intent and the confidence, and these two values will be used in design pattern selection logic.

4.3.2 Graphical user interface-web page

Consider a situation where a task is given to add logger functionality to the existing code base, simply this would be a Jira story or sub story which covers a unique requirement. This same example is used to explain the tool in next section as well.

The graphical user interface is a single page web application implemented using HTML, JavaScript, Joint.js and Bootstrap.

In the first section the user is asked to enter the problem description in a descriptive manner, including technical jargon and it is checked for the intent.



The screenshot shows a web form with two main sections. The first section is labeled 'Task Name:' and contains a text input field with the text 'Adding Logger class to the current application'. The second section is labeled 'Problem Description:' and contains a larger text area with the text 'There are two types of logger available based on the log destination. The logs shod be written either to a file or the console. Which log should be selected is mentioned in a property file and decided at runtime'. A small green circular icon with a white arrow is located in the bottom right corner of the text area. Below the text area is a blue button labeled 'Check Intent'.

Figure 4.8: Problem Description

A RSET api call to WA, is made from front end where the intent and the confidence taken from the WA response are set in the design pattern recommendation request.

Using the design panel, the user is asked to model the problem. In here proper understanding of the objects involved, their connection with other objects and purpose of each object should be identified properly.

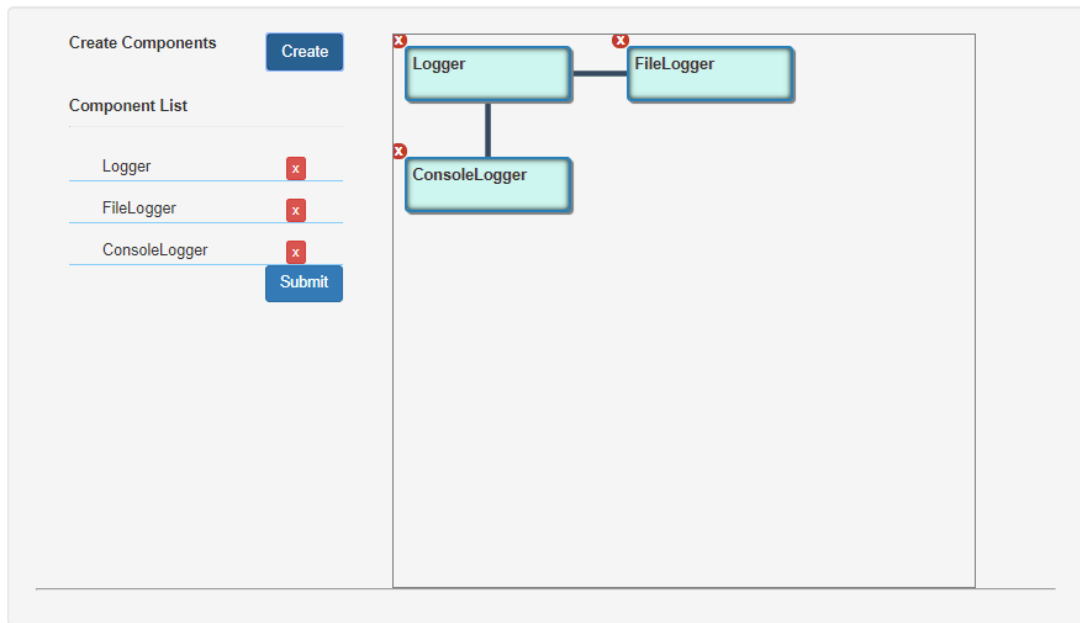


Figure 4.9: Design Panel

To create a component there is a button “Create”. When the button is clicked it will load a modal window. The modal window has several options to be selected by a user. These options help the user to identify the areas which need to be taken in to account when a system is designing. The options are related to the object’s relationships, object representation and the pattern selection criteria.

Create Component
✕

Component Name

Component Relationships

<input checked="" type="checkbox"/>	Is a	<input type="text" value="Components"/>	Selected Components
			• Logger ✕
<input type="checkbox"/>	Association	<input type="text" value="Components"/>	Selected Components
<input type="checkbox"/>	Aggregation	<input type="text" value="Components"/>	Selected Components
<input type="checkbox"/>	Composition	<input type="text" value="Components"/>	Selected Components

Areas to think

Object_structure

- Less number of constructor arguments
- Has incompatible Interface.
- Resolve incompatible interface.
- Required object aggregation to have the complete object.

Object_creation

Figure 4.10: Component creation model window

Once the problem modeling is done the submit button is and then the design pattern suggestion request is created and send via a REST api call to the backend.

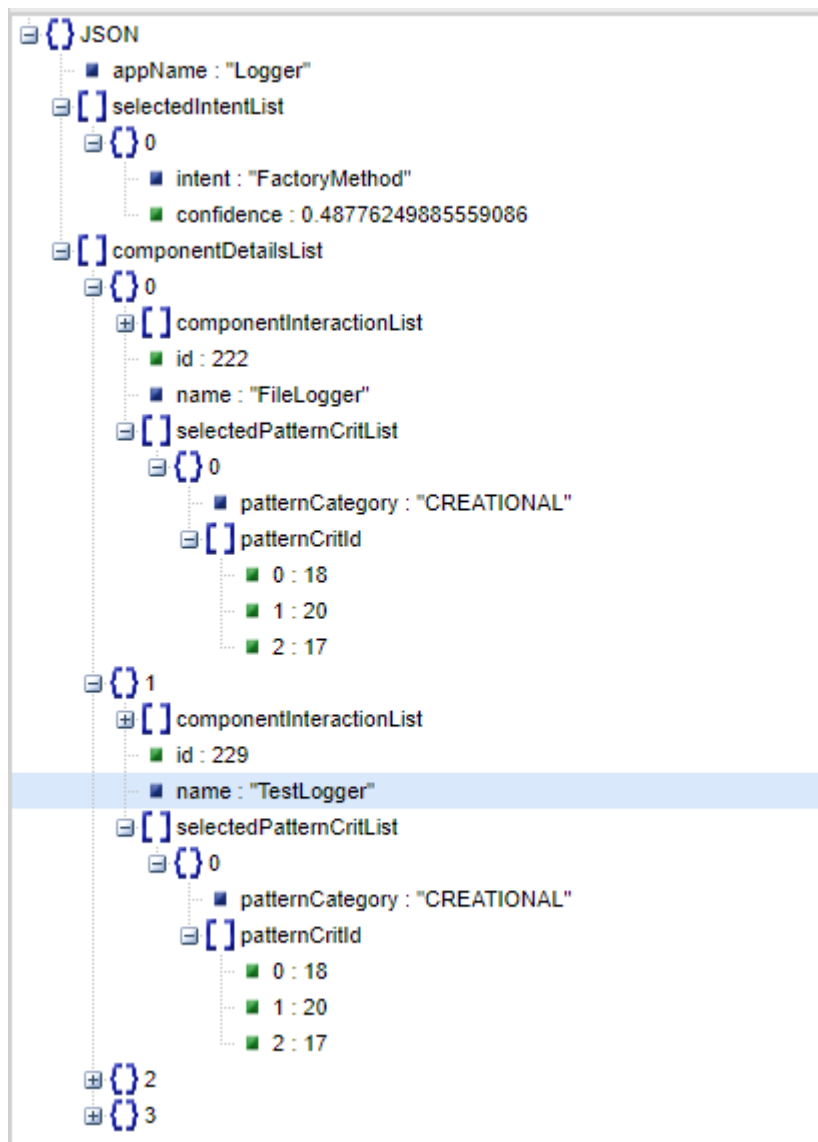


Figure 4.11: Design Pattern suggestion request

Design pattern reasoning and the generated UML class diagram are set in the Design pattern suggestion response. By reading the response UML diagram and the Design reasoning they are set in under the relevant div elements.

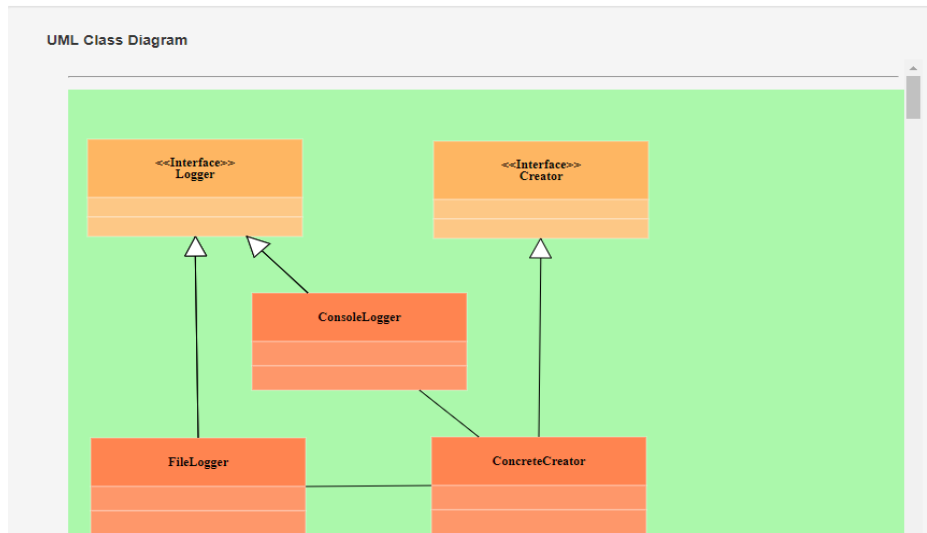


Figure 4.12: Generated UML diagram

Design pattern reasoning	
Pattern Explanation	<ul style="list-style-type: none"> As an example consider adding log function to current code base. The logs should be either written to a file or to the console. And which one to be used is mentioned in a property file. First thing that we need to consider is which classes will use this log function. In this scenario there is a high possibility of using log function by each and every class in the code base Then how to design the solution. In this case we need two classes and let's call them FileLogger and ConsoleLogger Both classes are used to log, therefore we can think of a super class called Logger FileLogger and ConsoleLogger are subclasses of the Logger and inherit the log() method. Now we have the two classes ready. But how to call them. Anyway we have to check the property file value to select which logger class to be selected. So are we going to implement the log class selection logic in each and every class and do the logger class instantiation. If we do so whenever we create a new class we have to define the log class selection logic there. Apart from that if we want to add another log class, let's say a XmlLogger, then we have to go each and every class and do the modifications. Which is not the correct way. If some code block is common then we can separate them into a class and refer to it. In this case we can define a class which is responsible for creating the correct log instance. Simply a log class creator Then we only need to modify that creator class. By calling the log creator class we can have the correct log instance.
Pattern name	:Factory Method
Pattern category	: CREATIONAL
Pattern intent	: Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses
Pattern applicabilities	<ul style="list-style-type: none"> A class can't anticipate the class of objects it must create. A class wants its subclasses to specify the objects it creates. Classes delegate responsibility to one of several helper subclasses, and you want to localize the knowledge of which helper subclass is the delegate.
Pattern consequences	<ul style="list-style-type: none"> Provides hooks for subclasses. Connects parallel class hierarchies.

Figure 4.13: Design reasoning

4.3.3 Design pattern selection framework service

The design pattern selection framework is a spring boot application which exposes REST apis. The main responsibility of this service is determining the best design pattern for user specific problem.

As the first step the design pattern suggestion request object is processed. For each component which is involved with the design, are evaluated with the selected pattern criteria. Each design pattern criteria are given a weight based on the support of representing the design pattern intent. Some criteria have different weights under different patterns. After considering weights of each criterion, design pattern with the highest score will be selected. At the same time if the WA suggested pattern has more than 0.75 of confidence it is selected as the best pattern. If the same component is eligible for more than one design pattern, then the second pattern is selected based on the related patterns.

Once the design pattern is selected, the user created component details will be used to determine the relevant classes in the design structure. If proper matching is found, then the default class values will be replaced. When the user design is closer to the correct design the quality of the class diagram will be improve. Simply the accuracy of the UML diagram depends on the user design.

After generating the required code segment related to the UML diagram, the other required details for design reasoning are queried from the database.

As the final step design pattern suggestion response is created using UML diagram details and the design reasoning details. The response is rendered appropriately in the front end and updates the relevant div elements.

```

JSON
  status : "SUCCESS"
  errorMessage : null
  patternName : "Factory Method"
  patternCategory : "CREATIONAL"
  patternIntent : "Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses"
  patternExplanation
    0 : "As a example consider adding log function to current code base. The logs should be either written to a file or to the console. And which one to be used is mentioned in a property file."
    1 : "First thing that we need to consider is which classes will use this log function. In this scenario there is a high possibility of using log function by each and every class in the code base"
    2 : "Then how to design the solution. In this case we need two classes and lets called them as FileLogger and ConsoleLogger"
    3 : "Both classes are used to log, therefor we can think of a super class called Logger"
    4 : "FileLogger and ConsoleLogger are subclasses of the Logger and inherit the log() method."
    5 : "Now we have the two classes ready. But how to call them. Anyway we have to check the property file value to select which logger class to be selected."
    6 : "So are we going to implement the log class selection logic in each and every class and do the logger class instantiation."
    7 : "If we do so whenever we create a new class we have to define the log class selection logic there."
    8 : "Apart from that if want to add another logclass, lets say a XmlLogger, then we have to go each and every class and do the modifications."
    9 : "Which is not the correct way. If some code block is common then we can separate them in to a class and refer to it. "
    10 : "In this case we can define a class which is responsible for creating the correct log instance. Simply a log class creator"
    11 : "Then we only need to modify that creator class."
    12 : "By calling the log creator class we can have the correct log instance."
    13 : "So we can use Factory Method to designing."
  applicabilityList
  consequencesList
  umlDiagramList

```

Figure 4.14: Design pattern suggestion response

4.3.4 Knowledge base

The knowledge base is consisted of the database and the WA json. The database has stored all the related data.

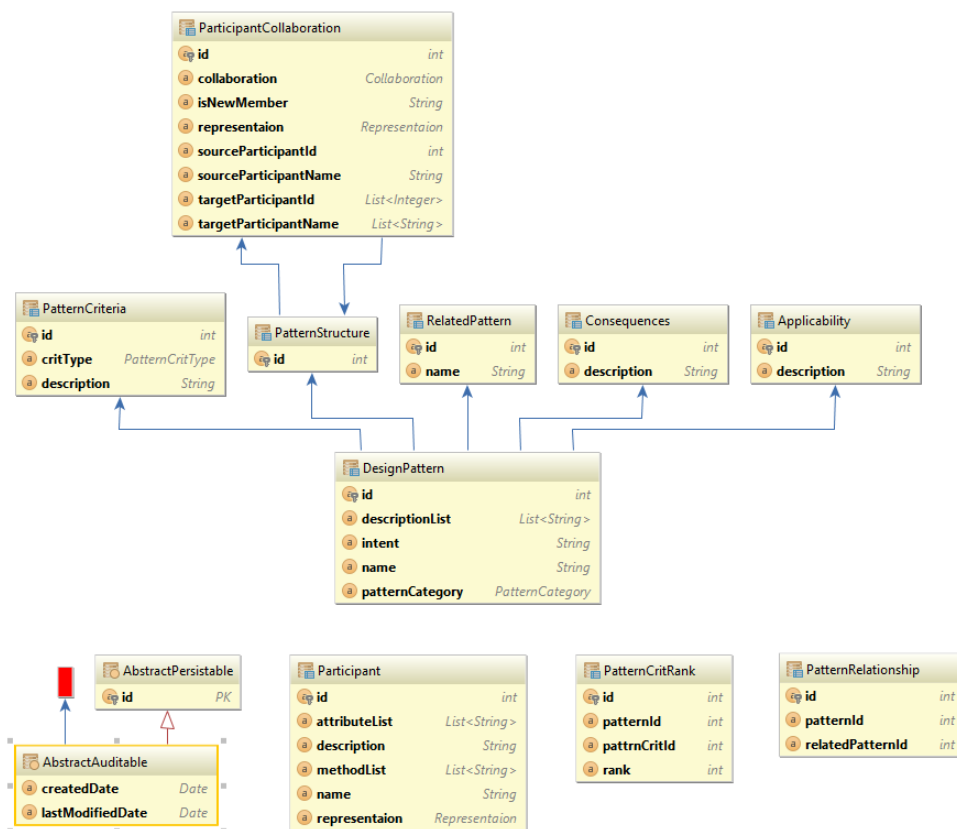


Figure 4.15: ER diagram of the database.

CHAPTER 5
DATA ANALYSIS AND EVALUATION

5.1 Introduction

To evaluate the developed prototype two surveys were conducted focusing IT professionals with different experience levels. These surveys were introduced to analyze and gather data on two different aspects.

First aspect was to identify the criteria that need to be fulfilled in order to learn design patterns and how to select them. The survey questions were to gather data on importance of the design patterns in solution designing, variations of knowledge on design patterns with the experience level, background knowledge required to understand the design pattern, background knowledge level they have and availability of reference resources to learn design pattern.

The second aspect was to evaluate the developed application in terms of usability, importance and expected learning outcome.

5.2 Analysis on adding design patterns

The IT professional who have participated in this survey belonged to different designation levels in the industry. Visual representation of designations distribution of the participants is depicted in figure 5.1.

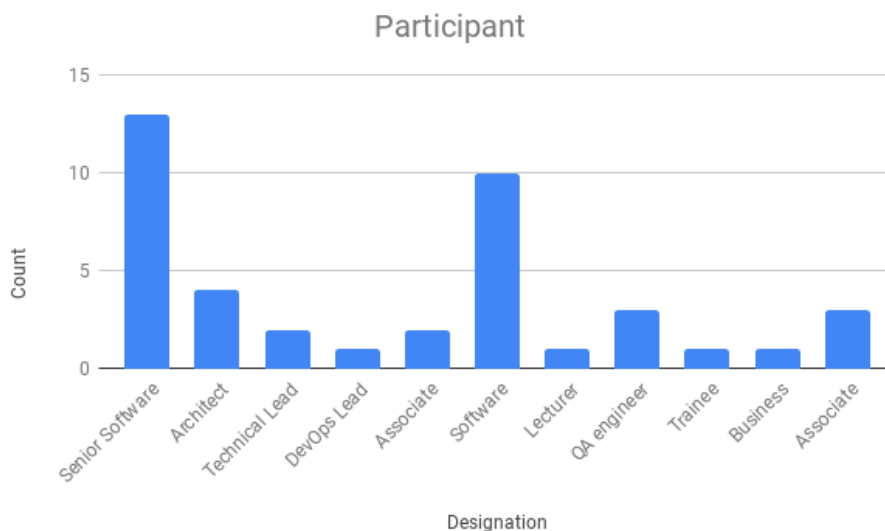


Figure 5.1: Participant count with respect to designation

Next focus was to identify importance of adding design pattern to the solution design in order to improve the quality of the design. According to the survey data 48.8% strongly agreed and 43.9% agreed. In total 92.7% agreed on adding design pattern would improve the quality of the program solution design.

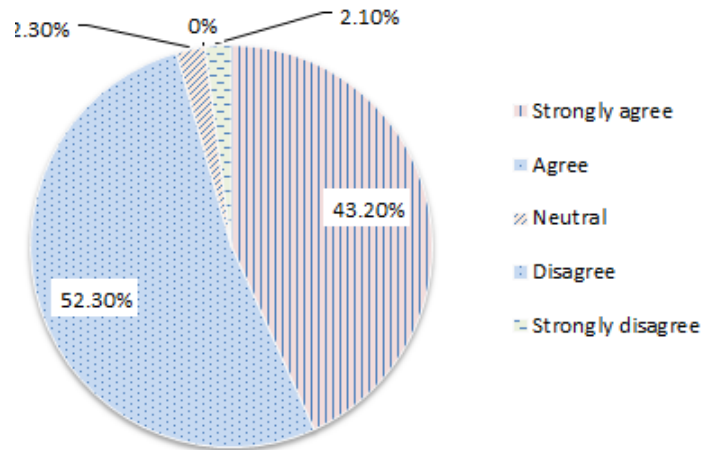


Figure 5.2: Importance of adding design patterns to the program solution design

Even though it is important to introduce the design patterns to the current design, the knowledge/experience on design patterns of IT professionals did not meet a significant level. The percentage of knowing more than 6 design patterns was 29.3%.

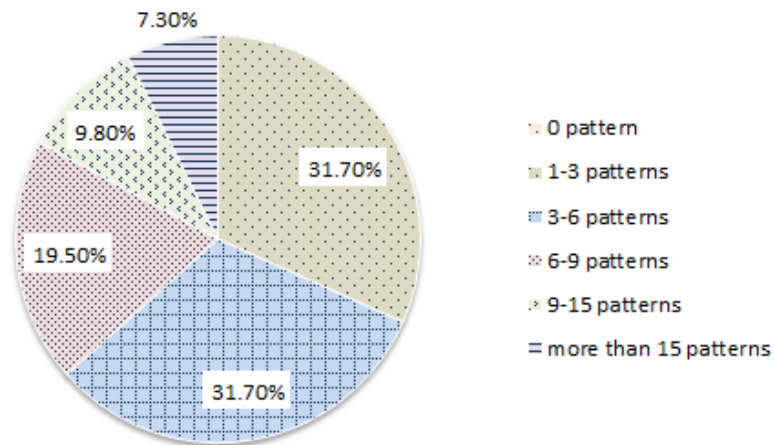


Figure 5.3: Experience in applying design patterns

5.3 Evaluation on design pattern selection criteria

In order to identify the matching design pattern of a scenario, the problem scenario should be analyzed first. To analyze, it is required to identify the aspect which needs to be considered. Simplified version of possible aspects was defined as the pattern selection criteria. The applicability of the defined criteria was evaluated with the participant responses.

Pattern Selection criteria usage

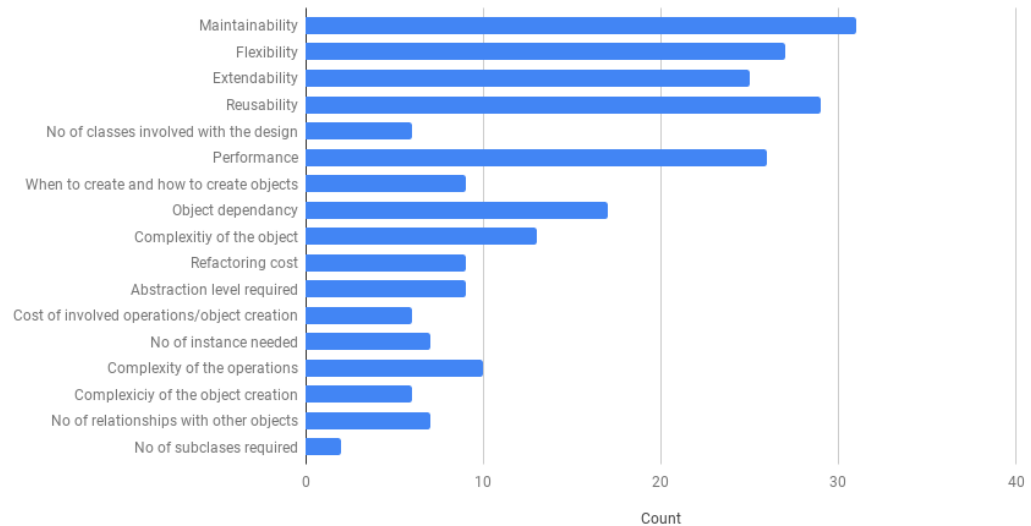


Figure 5.4: Pattern selection criteria usage

5.4 Evaluation on developed sample prototype application

The implemented sample prototype is evaluated in several ways and under different evaluation criteria,

Evaluation scenarios

1. Different problem scenarios
2. Same problem scenario with different users

Evaluation criteria

1. WA – selected pattern(intent) and confidence
 - a. Accuracy
2. Suggested design pattern
 - a. Accuracy
3. Generated diagram

- a. Accuracy
- b. Completeness
- 4. Design reasoning
 - a. Understandability
 - b. Clarity
- 5. Usability

To do the evaluation, the set of problems were selected which were used to explain the design patterns by experienced personal.

5.4.1 Evaluation of different problem scenarios

The set of problems considered are given in the table below.

Table 5.1: Different problem scenario

No	Problem scenario	Expected Design pattern
1	Create a Database connection class.	Singleton
2	Introduce logger to the existing system. There should be two loggers called ConsoleLogger and FileLogger. Which logger to be selected is mentioned in a property file.	Factory Method
3	Convert PDF document to word document. For the simplicity assume that conversion process needs only text, image, and table conversions.	Builder
4	The ABC system is using AWS services. But now they need to move to the similar service in Azure. Even though request format is somewhat similar it cannot call them directly due to incompatibilities.	Adapter
5	In an airport there are limited tracks which are available for landing. Assume that number of flights can come to the airport is greater than the number of available tracks. Communications	Mediator

	with the pilots are possible and need to develop a system to avoid the conflicts that can be occurred at flight landing.	
6	Create a remote controller to control both Fan and the AC. Fan and AC are separated systems.	Façade

Evaluation results are shown in the below table.

Table 5.2: Evaluation Results on different problem scenarios

Problem no	intent	Confidence	Selected pattern	Class diagram completeness
1	Prototype	0.87	Not suggested	No diagram
2	FactoryMethod	0.48	Factory Method	Complete
3	Builder	0.75	Builder	Complete
4	Adapter	0.26	Adapter	Default
5	Mediator	0.43	Mediator	Incomplete
6	Facade	0.53	Facade	Default

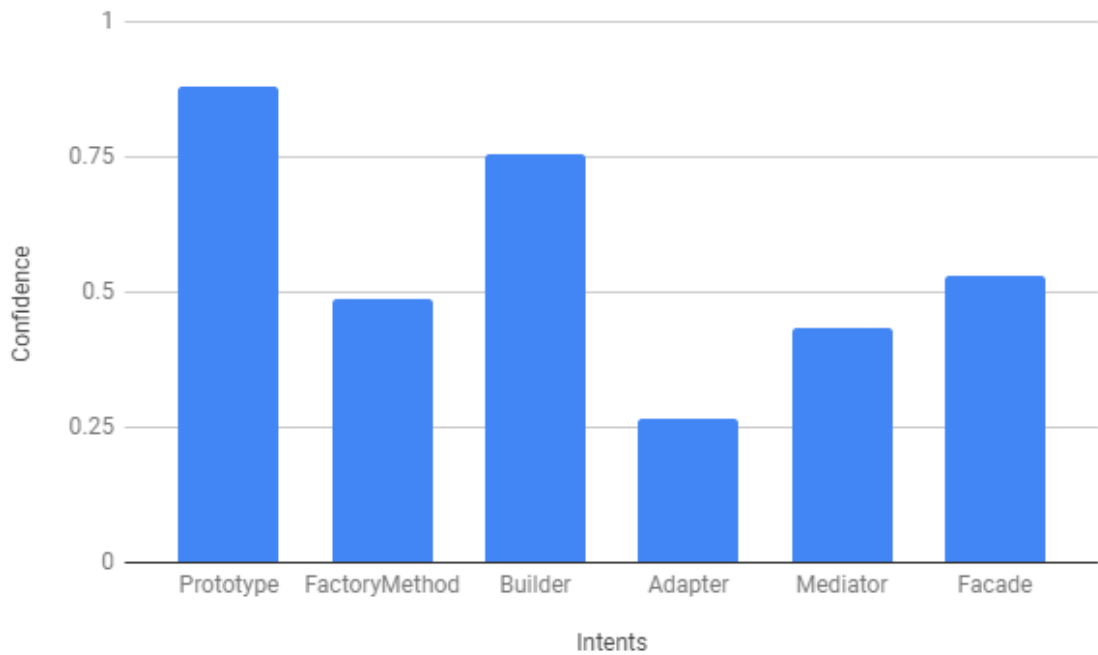


Figure 5.5: Watson assistant selected intents and confidence

Even though the intent was suggested by the WA, some of the intent was not matched with expected result which is the expected design pattern. According to figure 5.6, 83.3% was identified correctly and 16.7% was identified incorrectly.

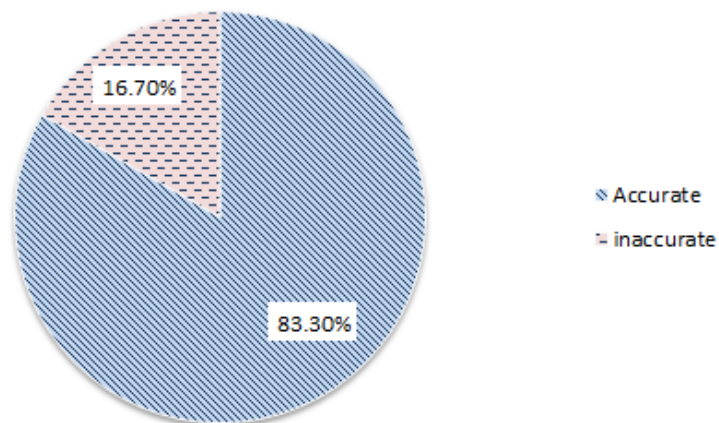


Figure 5.6: Accuracy of the returned intent from Watson Assistant

The design pattern suggestion frame work performed at 83.3% accuracy in identifying the correct pattern.

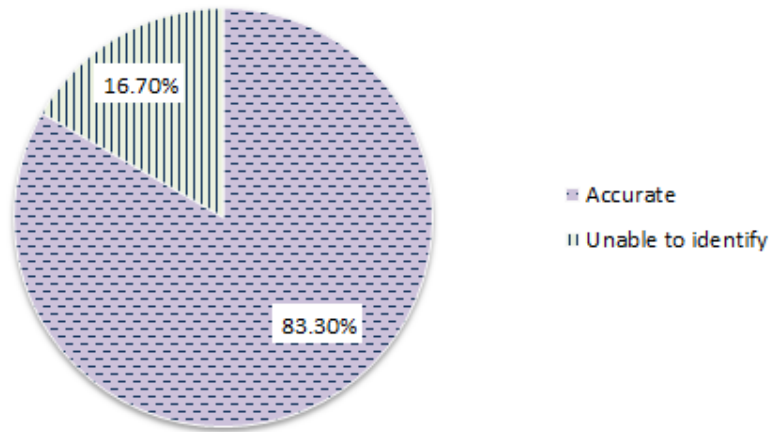


Figure 5.7: Accuracy of the design pattern suggestion frame work

When a design pattern was suggested, the quality of the generated class diagram was evaluated as well. The generated class diagram can be in one of these states, complete, default, incomplete, not generated, and incorrect. Complete would be, if the generated diagram is correct and user defined components are also added to the diagram in a correct way. Default is referred to the default diagram where the generic diagram is provided for the design pattern. Incomplete means that the diagram is generated but some links, classes are missing. When the diagram is not related with the pattern or not supported with the default structure then it is considered as incorrect. The complete diagram was generated with the percentage of 33.3%.

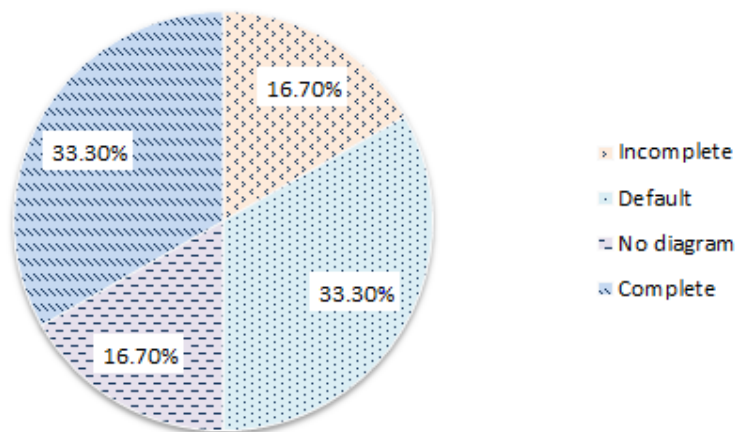


Figure 5.8 Completeness of the generated class diagrams

5.4.2 Evaluation on same scenario with different users

The same problem scenario was tested with different users. The main purpose of testing with the same scenario was to check the different ways of understanding the same scenario, different ways of designing the solution and the pattern criteria selected. After modeling the problem solution, user feedback was collected in the form of a survey.

Usability of the developed prototype, learning experience, quality of the reasoning provided and value added to the knowledge on design pattern were measured using the survey.

According to the survey 38.5% of users were able to use the application at ease, 23.1% were neutral 30.8% of users found it hard to use it at the beginning.

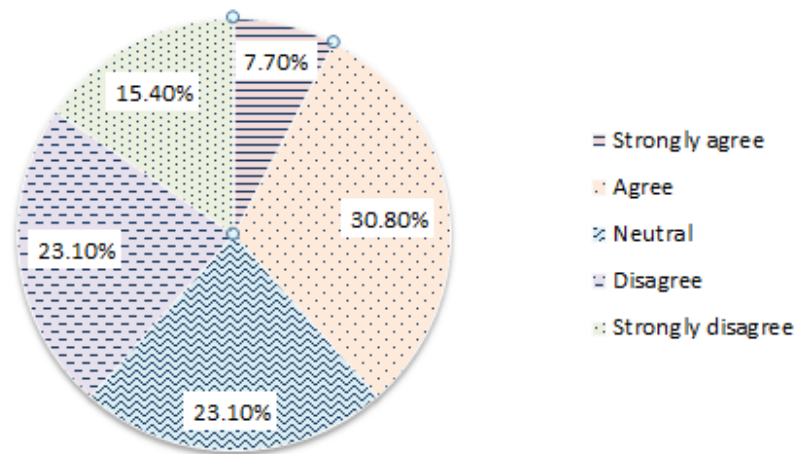


Figure 5.9: Usability of the developed application

Since the all the input should be given by the user, the user should think before they proceed with modeling the solution. It makes the user interaction with the application strong. 69.3% of users found that the application is more interacted with the users and 30.8% were found it as neutral.

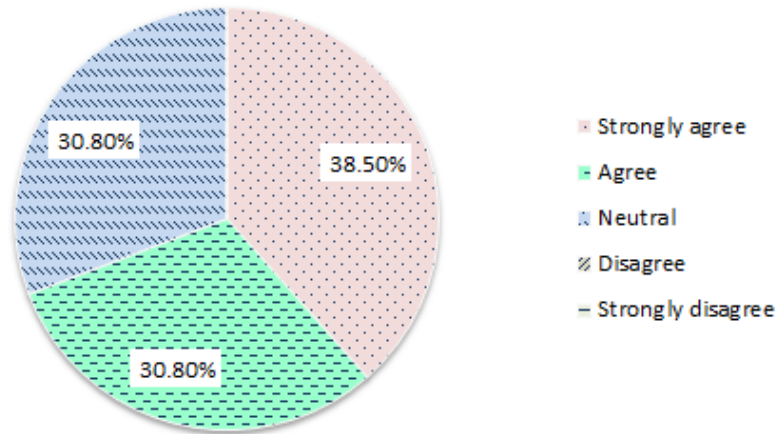


Figure 5.10: User interaction involved in the form of thinking and analyzing the given scenario

The learning experience gained through the application can be calculated as 84.8%. The design pattern criteria provided the users with a different learning experience.

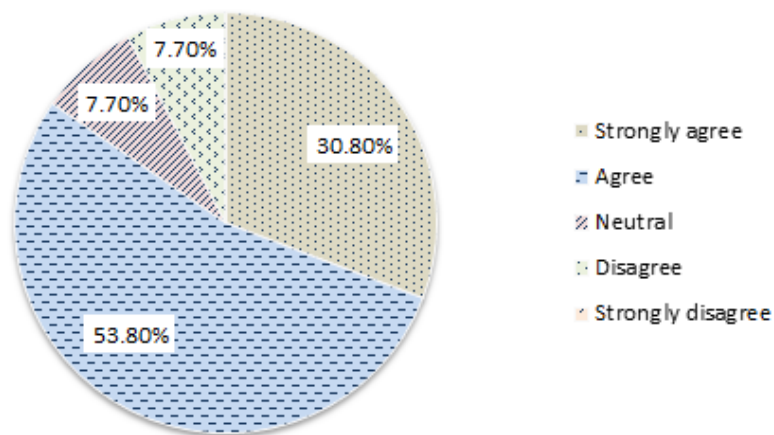


Figure 5.11: Learning experience with identifying criteria to think when doing a solution design

From the users involved in the survey, 61.5% of users had found the pattern selection reasoning provided was understandable enough to grab the idea behind it.

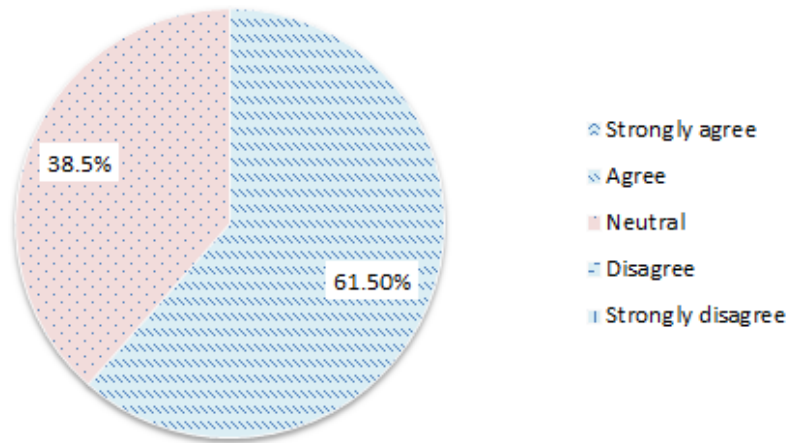


Figure 5.12: Understandability of the generated pattern selection reasoning

The percentage of users who have accepted that the overall process gained them more knowledge is 92.3%.

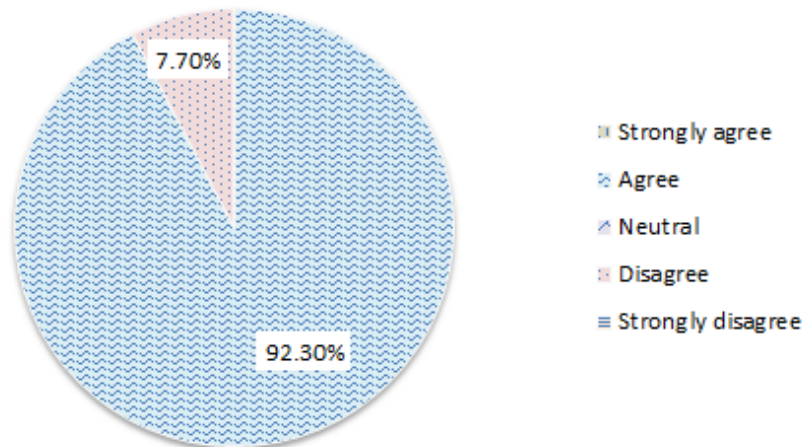


Figure 5.13: Value addition to the knowledge

The developed application had been identified as a design pattern learning tool by 69.3% of users who participated in the survey.

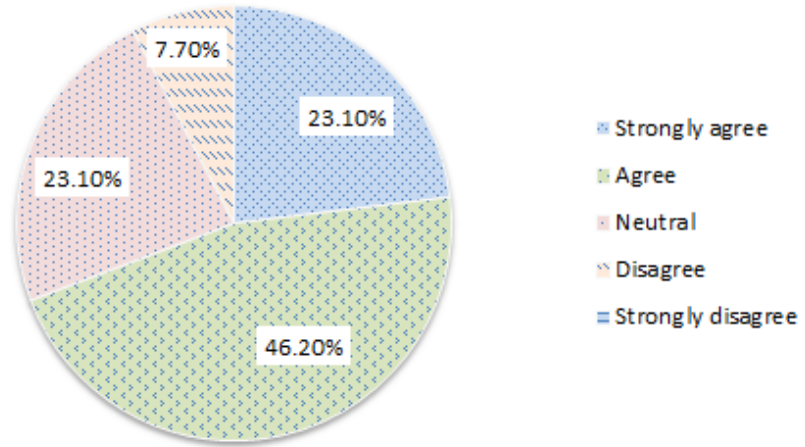


Figure 5.14: Developed application as a design pattern learning tool

CHAPTER 6
CONCLUSION

Designing phase of a software application is much more critical and requires high attention, because it decides the future of the software application. If the application is developed based on a quality design, then it reduces the future hassles more likely to happen when maintaining the application, adding extensions and doing modifications. If the design can include the expert experience which is achieved through developing similar applications, then possible future burdens can be reduced. Usage of design pattern in a proper way helps to add expert experience to the application design.

But the problem is most of the developers do not have required knowledge on design pattern. Even though there are thousands of resources available on the internet, sometimes it is difficult to understand design patterns for novice developer.

6.1 Research contribution

The research mainly focuses on defining a frame work to select best matching design pattern by analyzing the problem scenario. The main difference in this research is the framework asks user to do the analysis of the scenario and give input accordingly. The reason why the analysis part is given to the user is, then the user will be able to learn how to select a pattern and what needs to be considered when designing a solution. It will help to improve their knowledge and analytical skills required in designing an application.

Most important part of the design pattern selection framework is the set of criteria defined for each pattern to have identified them uniquely. The defined pattern criteria should be simple enough to be identified by the users and should be specific enough to identify a candidate design pattern.

The suggested design pattern selection tool is more user interactive. It asks the user to design the solution by considering the purpose of each object involved with the design, and how to form them together.

The application provides a UML diagram if any pattern suggestion is found. This gives a chance to the user to compare own design and the generated UML diagram, which will help user to find his mistake if there are any. Under design pattern

selection reasoning it describes why the pattern is selected and the UML structure using an example. This will further help the user to understand the situation and the pattern applicability as well.

6.2 Research Limitation

The main limitations in this framework are, it can only identify 23 design patterns, and the current application is able to identify possible one design pattern at a time for a given problem scenario. This is focused on object-oriented programming. The pattern selection has strong relationships with the defined pattern selection criteria list, if user select them without thinking then it will produce incorrect results.

To use this application, the user should have development experience and knowledge on object-oriented concepts and object-oriented principles.

6.3 Future Work and conclusion

The suggested user interactive design pattern selection framework is developed as a sample application only covering the happy path. The application should cover the negative scenarios as well.

The logic is implemented to find a candidate design pattern for a given scenario, it should be modified to suggest multiple design patterns if there any. Along with the multiple design pattern suggestion, the UML class generation process should be changed to cater more than one design pattern. The UI should be improved to use the application easily.

The framework should allow adding new patterns and it needs to provide an interface to add them.

The created skill set in Watson Assistant should be trained further to identify the intent correctly, and latterly it can be added in a form of chat bot to the application in order to get more attraction with the user and get expected user inputs. The chat bot can be defined as a virtual teacher.

Adding design pattern to the design would improve the quality of the design, but the IT professionals has less knowledge on design patterns. To understand design

patterns properly, the basic understanding on object-oriented concepts and object-oriented principles are required. Along with this and the guidance of an experienced developer a high-quality application can be developed enriched with proper design patterns.

References

- [1] E. Gamma, R. Helm, R. Johnson and J. Vlissides, "Design Patterns: Elements of Reusable Software," Addison-Wesley, 1994.
- [2] T. Erl, "SOA Design Patterns," Prentice Hall, New York, 2008.
- [3] J. Knox, "Adopting Software Design Patterns in an IT Organization: An Enterprise Approach to Add Operational Efficiencies and Strategic Benefits," M.S. thesis, AIM program, Dept of Computer and Information Science, University of Oregon, 2011
- [4] I. Sommerville, "Software Engineering," Addison-Wesley, Boston, 2004.
- [5] D.K. Kim, C. Khawand, C. el, "An approach to precisely specifying the problem domain of design patterns," *Journal of Visual Languages and Computing*, vol.18, no.6, pp. 560-591, 2007.
- [6] W. Zimmer, "Relationships between Design Patterns," *Proceedings of the First Conference on Pattern Languages and Programming*, Addison-Wesley, 1994.
- [7] J. Noble, "Classifying Relationships between Object-Oriented Design Patterns," *Australian Software Engineering Conference (ASWEC)*, pp.98-107, USA, 1998.
- [8] J. Vlissides, "Pattern Hatching Composite Design Patterns (They Aren't What You Think)," C++ Report, June 1998.
- [9] B. William, M. McNatt James, Bieman, "Coupling of Design Patterns: Common Practices and Their Benefits," *Computer Software & Applications Conf. (COMPSAC 2001)*, USA, October 2001.

- [10] M. Hills, P. Klint, T.V.D Storm, and J. Vinju, "A Case of Visitor versus Interpreter Pattern," *Proceedings of the 49th international conference on Objects, models, components, patterns*, pp.228-243, Switzerland, June 28 - 30, 2011.
- [11] R. Bala, K. K. Kaswan, "Strategy Design Pattern," *International Journal of Science and Research*, vol.3, no.8, August 2014.
- [12] A.Meiapane, J.Prabavathi, V. Prasanavenkatesan, "Strategy pattern: payment pattern for internet banking", *International Journal of Information Technology and Engineering (IJITE)*, March 2012.
- [13] M. D. Parsana, J. N. Rathod and J. D. Joshi, "Using Factory Design Pattern for Database Connection and Daos (Data Access Objects) With Struts Framework," *International Journal of Engineering Research and Development*, vol.5, no.6, pp.39-47, December 2012.
- [14] M. R. J. Qureshi and W. Al-Geshari , "Proposed Automated Framework to Select Suitable Design Pattern," *International Journal of Modern Education and Computer Science*, vol.9, no.5, pp.43- 49 , 2017
- [15] F. Palma, H. Farzin, Y.G Guéhéneuc and N Moha, "Recommendation System for Design Patterns in Software Development: An DPR Overview," *Recommendation Systems for Software Engineering (RSSE)*, 2012.
- [16] S. S. Suresh, M. M. Naidu and S. A. Kiran, "Design Pattern Recommendation System (Methodology, Data Model and Algorithms) ," *International Conference on Computational Techniques and Artificial Intelligence*, 2011
- [17] I. Issaoui, N. Bouassida, and H. Ben-Abdallah, "A New Approach for Interactive Design Pattern Recommendation," *Lecture Notes on Software Engineering* vol. 3, no. 3, pp. 173-178, 2015.

- [18] E. M. Sahly and O. M. Sallabi, "Design pattern selection: A solution strategy method," *International Conference on Computer Systems and Industrial Informatics*, Sharjah, UAE, pp.1-6, 2012.
- [19] E. M. Salah, M. T. Zabata, and O. M. Sallabi. "Dps: Overview of design pattern selection based on mas technology," *Distributed Computing and Artificial Intelligence*, Springer, Cham, pp.243-250, 2013.
- [20] W. Muangon and S. Intakosum, "Case-based Reasoning for Design Patterns Searching System," *International Journal of Computer Applications*, vol. 70, no. 26, pp. 16–24, 2013.
- [21] P. Kuchana, *Software Architecture Design Patterns in Java*. 2nd ed. Auerbach Publications. 2004
- [22] E. Freeman, E. Robson, B. Bates and K. Sierra, *Head first design patterns*. O'Reilly Media, Inc. 2004.
- [23] "Design Patterns," [Online]. Available: https://sourcemaking.com/design_patterns . [Accessed Dec.12,2018].
- [24] Baeldung, "Introduction to Creational Design Patterns," Nov. 27, 2018. [Online]. Available: <https://www.baeldung.com/creational-design-patterns>. [Accessed Dec.12.2018].
- [25] Jonathan Aldrich,"Design Patterns," 2011. [Online]. Available : <https://www.cs.cmu.edu/~aldrich/courses/15-214-12fa/slides/10-12-design-patterns.pdf>. [Accessed Dec.24, 2018].
- [26] "Dessign Patterns," [Online]. Available: <https://www.oodeesign.com/>. [Accessed Jan .10, 2019].

[27] “Design Patterns in Java,” [Online]. Available:
<https://www.javatpoint.com/design-patterns-in-java>. [Accessed Jan .15, 2019].

[28] “Software Design Patterns,” [Online]. Available:
<https://www.geeksforgeeks.org/software-design-patterns/>. [Accessed Jan .18, 2019].

[29] D. Banas. Design Patterns Video Tutorial (Aug .19, 2012). Accessed:
Feb. 2, 2019. [Online Video]. Available:
https://www.youtube.com/watch?v=vNHpsC5ng_E&list=PLF206E906175C7E07

[30] C. Okhravi. Design Patterns in Object Oriented Programming (Dec. 28, 2016).
Accessed: Feb. 12, 2019. [Online Video]. Available:
<https://www.youtube.com/playlist?list=PLrhzvIcii6GNjpARdnO4ueTUAVR9eMBpc>.

APPENDIX A: Survey Questioner 01

Section 2 of 6

✕ ⋮

User details

Description (optional)

My job role is *

- Trainee developer
- Associate Software Engineer
- Software Engineer
- Senior Software Engineer
- Associate Technical Lead
- Technical Lead
- Architect
- Other...

I have experience of in IT industry *

- 0-1 year
- 1-2 years
- 2-5 years
- 5-8 years
- 8-10 years
- more than 10 years

Importance of the design patterns

Description (optional)

Do you think that design patterns are important? *

- Strongly agree
- Agree
- Neutral
- Disagree
- Strongly disagree

Do you think that design patterns can improve the quality of the program design? *

- Strongly agree
- Agree
- Neutral
- Disagree
- Strongly disagree

Learning design patterns

Description (optional)

I can understand design patterns easily *

- Strongly agree
- Agree
- Neutral
- Disagree
- Strongly disagree

I can easily find resources to learn about design patterns *

- Strongly agree
- Agree
- Neutral
- Disagree
- Strongly disagree

It is important to have knowledge on OOP concepts to understand design patterns *

- Strongly agree
- Agree
- Neutral
- Disagree
- Strongly disagree

It is important to have knowledge on OOP principles to understand design patterns *

- Strongly agree
- Agree
- Neutral
- Disagree
- Strongly disagree

Coupling and cohesion have strong relationship with design patterns

Multiple choice

- Strongly agree
- Agree
- Neutral
- Disagree
- Strongly disagree



Required



Knowledge/Experience on design patterns

Description (optional)

My knowledge on OOP concepts is *

	1	2	3	4	5	
Very Weak	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Very Good

My knowledge on OOP principles is *

	1	2	3	4	5	
Very Weak	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Very Good

My knowledge on cohesion and coupling is *

	1	2	3	4	5	
Very Weak	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Very Good

I know aboutdesign patterns *

- 0
- 1 - 3
- 3 - 6
- 6 - 9
- 9 - 15
- more than 15

I have the practical experience on design patterns *

- 0
- 1 - 3
- 3 - 6
- 6 - 9
- 9 - 15
- more than 15

The design patterns I have used are

- Singleton
- Factory Method
- Builder
- Prototype
- Abstarct Factory
- Adapter
- Bridge
- Decorator
- Flyweight
- Facade
- Chain of responsibility
- Visitor
- Interpreter
- Iterator
- State
- Stratergy
- Command
- Momento
- Mediator
- Proxy
- Template Method
- Composite
- Observer

How to select a design pattern

Description (optional)

When Selecting a design pattern I consider about

- Maintainability
- Extendability
- Flexibility
- Performance
- Refactoring cost
- Reusability
- Abstraction level required
- Cost of involved operations/object creation
- No of instance needed
- When to create and how to create objects
- Complexity of the object
- Complexity of the operations
- Complexity of the object creation
- No of relationships with other objects
- Object dependancy
- No of subclasses required
- No of classes involved with the design

If there are more criteria to be considered before selecting a design pattern, please specify

Long answer text

APPENDIX B: Survey Questioner 02

Section 2 of 2



User feedback

Description (optional)

I was able to use the tool easily *

- Strongly Agree
- Agree
- Neutral
- Disagree
- Strongly Disagree

When I use the tool I had to think first and follow the steps *

- Strongly Agree
- Agree
- Neutral
- Disagree
- Strongly Disagree

When I use the tool I got to know what are the aspects need to be considered *
when design a problem solution

- Strongly Agree
- Agree
- Neutral
- Disagree
- Strongly Disagree

The options given in the tool is easy to understand *

- Strongly Agree
- Agree
- Neutral
- Disagree
- Strongly Disagree

The reason for the design pattern recommendation was understandable *

- Strongly Agree
- Agree
- Neutral
- Disagree
- Strongly Disagree

I feel I have learnt something on design pattern while using the tool *

- Strongly Agree
- Agree
- Neutral
- Disagree
- Strongly Disagree

I have learnt one design pattern and applicability of it with the provided example *

- Strongly Agree
- Agree
- Neutral
- Disagree
- Strongly Disagree

The tool can be used to learn design pattern *

- Strongly Agree
- Agree
- Neutral
- Disagree
- Strongly Disagree