

**A CONSOLIDATED APPROACH FOR  
BUILDING A SOFTWARE DESIGN  
PATTERN RECOMMENDATION SYSTEM**



DİLARA BOZOKLAR

JUNE 2019

**A CONSOLIDATED APPROACH FOR  
BUILDING A SOFTWARE DESIGN  
PATTERN RECOMMENDATION SYSTEM**

A THESIS SUBMITTED TO  
THE GRADUATE SCHOOL OF  
NATURAL AND APPLIED SCIENCES OF  
IZMIR UNIVERSITY OF ECONOMICS

BY

**DİLARA BOZOKLAR**

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

IN THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES


JUNE 2019

## M.S. THESIS EXAMINATION RESULT FORM


Approval of the Graduate School of Natural and Applied Sciences

  
Prof. Dr. Abbas Kenan Çiftçi  
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

  
Assoc. Prof. Dr. Süleyman Kondakçı  
Head of Department


We have read the thesis entitled “**A Consolidated Approach for Building a Software Design Pattern Recommendation System**” completed by **DİLARA BOZOKLAR** under supervision of **Asst. Prof. Dr. Ufuk Çelikkan** and we certify that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

  
Asst. Prof. Dr. Ufuk Çelikkan  
Supervisor

### Examining Committee Members

Date: 24.06.2019


Asst. Prof. Dr. Ufuk Çelikkan  
Dept. of Software Engineering, IUE



Assoc. Prof. Senem Kumova Metin  
Dept. of Software Engineering, IUE



Asst. Prof. Dr. Umut Avcı  
Dept. of Software Engineering, Yaşar University



## ABSTRACT

# A CONSOLIDATED APPROACH FOR BUILDING A SOFTWARE DESIGN PATTERN RECOMMENDATION SYSTEM

DİLARA BOZOKLAR

M.S. in Computer Engineering

Graduate School of Natural and Applied Sciences

Supervisor: Asst. Prof. Dr. Ufuk Çelikkan

June 2019

Software design patterns are standard solutions to common problems found in software development and design. Among many other benefits that they offer, they enable the creation of reusable, extensible and easy to maintain software systems. However, the complexity of comprehending design patterns hinders software designer's ability to adapt software design patterns in software design and development. For novice developers choosing the right pattern for a given design context and situation becomes a challenging task. For this reason, a design pattern recommendation system can be of great help especially to the novice designers and developers to instantiate the right pattern in their design problems. In this thesis, we propose a consolidated approach by combining text based information retrieval, case based recommendation and question based recommendation to suggest an adequate pattern to a design problem. A tool has been implemented using this methodology. It is an interactive tool which first automatically recommends a design pattern by using text based information retrieval and case based recommendation. It then improves the results after a question / answer session. The recommended patterns are ranked and presented to the user as a list of alternatives. The effectiveness of the tool was tested on several scenarios used in a design pattern course. Our preliminary evaluation shows that in majority of the cases, the correct design pattern is placed in the top three.

*Keywords:* Design Patterns, Interactive Recommendation System, Information Retrieval, Document Similarity, Case Based Recommendation.

## ÖZ

# YAZILIM TASARIM ŞABLONLARI TAVSİYE EDEN BİR SİSTEMİN OLUŞTURULMASI İÇİN BİRLEŞİK BİR YAKLAŞIM

DİLARA BOZOKLAR

Bilgisayar Mühendisliği, Yüksek Lisans

Fen Bilimleri Enstitüsü

Tez Danışmanı: Dr. Öğr. Üyesi Ufuk Çelikkan

Haziran 2019

Yazılım tasarım şablonları, yazılım geliştirme ve tasarımda sık karşılaşılan sorunlara standart çözümler sağlarlar. Sundukları birçok fayda arasında, yeniden kullanılabilir, genişletilebilir ve bakımı kolay yazılım sistemlerinin geliştirilmesine olanak tanıyor olmaları vardır. Tasarım şablonlarının anlaşılmasının karmaşıklığı, bu şablonlarının tasarımcılar tarafından yazılım tasarımı ve geliştirilmesine uyarlanmasını zorlaştırmaktadır. Tecrübesiz geliştiriciler için, belirli bir tasarım bağlamında doğru modeli seçmek zorlu bir görev haline gelir. Bu nedenle, tasarım şablonları öneren sistemler, özellikle tecrübesiz tasarımcılara ve geliştiricilere, tasarım problemlerinde doğru şablonu örnekleme konusunda yardımcı olabilir. Bu tezde, metin tabanlı bilgi alma, vaka bazlı öneri ve soru bazlı öneri metotlarını birleştirip, uygun bir şablon önerecek bütünsel bir yaklaşım önerilmektedir. Bahsedilen bu yaklaşımı kullanarak bir uygulama geliştirilmiştir. Bu etkileşimli bir uygulama olup, ilk önce metin tabanlı bilgi alımı ve vaka tabanlı öneriyi kullanarak otomatik olarak bir tasarım şablonu önerir. Bunu takip eden soru/cevap oturumu sayesinde sonuçlar iyileştirilir. Elde edilen sonuçlar sıralanır ve kullanıcıya sunulur. Uygulamanın etkinliği, bir tasarım deseni dersinde kullanılan çeşitli senaryolar kullanılarak test edilmiştir. Ön değerlendirmemiz, çoğu durumda doğru tasarım şablonunun ilk üçe yerleştirildiğini göstermektedir.

*Anahtar Kelimeler:* Tasarım Şablonları, Etkileşimli Öneri Sistemi, Bilgi Elde Etme, Belge Benzerliği, Vaka Tabanlı Öneri.

## ACKNOWLEDGEMENT

I would like to express my special thanks of gratitude to my advisor Asst. Prof. Dr. Ufuk elikkan who gave me support while writing this thesis. Thank you for your valuable advice, patience, encouragement and efforts to expand the scope of this research.

Also, I would like to thank my all friends who are with me in every situation, especially Meltem Ergin who supported me throughout the entire process. Thank you for everything.

Finally, I must express my very profound gratitude to my family; Galip Bozoklar, Fatma Bozoklar, Bekir Bozoklar and Dođacan Tuđrul for providing me with unfailing support and continuous encouragement throughout my study and through the process of writing this thesis. Their love and help encouraged me to do new things in my life. Their presence, whenever I need, always made me feel strong against any challenge. I'm so lucky and blessed to have such amazing people in my life. Thank you.

# TABLE OF CONTENTS

<b>Front Matter</b>	<b>i</b>
Abstract . . . . .	iii
Öz . . . . .	iv
Acknowledgement . . . . .	v
Table of Contents . . . . .	vi
List of Tables . . . . .	vii
List of Figures . . . . .	viii
<b>1 Introduction</b>	<b>1</b>
<b>2 Literature Review</b>	<b>4</b>
2.1 Text Based Recommendation . . . . .	5
2.2 Case Based Recommendation . . . . .	5
2.3 Question Based Recommendation . . . . .	6
2.4 Combined Approaches . . . . .	8
2.5 Miscellaneous Approaches . . . . .	8
<b>3 Methodology</b>	<b>10</b>
3.1 Text Based Recommendation . . . . .	12
3.1.1 Data Collection and Assembly . . . . .	12
3.1.2 Text Preprocessing . . . . .	14
3.1.3 Tokenization and Normalization . . . . .	14

3.1.4 Feature Extraction, Term Weighting and Vector Space Model . . . . . 17

3.1.5 n-gram Features . . . . . 19

3.1.6 TF-IDF . . . . . 20

3.1.7 Cosine Similarity . . . . . 21

3.2 Case Based Recommendation . . . . . 22

3.3 Question Based Recommendation . . . . . 23

3.3.1 Design Pattern Methodology Example . . . . . 28

**4 Implementation and Results 32**

**5 Conclusion 45**

**A Design Patterns Scenarios 48**

**B Recommendation Tool GUI 54**



## LIST OF TABLES

3.1	Number of Scenarios for Each Software Design Pattern . . . . .	13
3.2	Bag of Words . . . . .	18
3.3	Extracting n-grams for the Adapter Pattern . . . . .	19
3.4	Questions for Determining Knowledge Level . . . . .	24
3.5	Pattern Questions . . . . .	26
3.6	Text Based Recommendation Results for Scenario #24 . . . . .	29
3.7	Text and Case Based Recommendation Results for Scenario #24 . . . . .	29
3.8	Specifying Knowledge Level . . . . .	30
3.9	QSM Results for Scenario #24 for the Expert Designer . . . . .	30
3.10	Text, Case and Question Based Recommendation Results for Scenario #24 . . . . .	31
3.11	Summary of Progress in Rankings . . . . .	31
4.1	The Effect of Using Wikipedia Sources on Scenairo #1 . . . . .	33
4.2	Text Based Recommendation Results . . . . .	34
4.3	Text Based Recommendation Results Confusion Matrix . . . . .	35
4.4	Text and Case Based Recommendation Results . . . . .	36
4.5	Scenario #18 Results . . . . .	36
4.6	Scenario #36 Results . . . . .	38
4.7	Scenario #7 Results . . . . .	39
4.8	Scenario #20 Results . . . . .	40
4.9	Scenario #36 Final Rankings for Novice Designer . . . . .	41

*LIST OF TABLES*

4.10 Scenario #7 Final Rankings for Intermediate Designer . . . . . 42

4.11 Scenario #20 Final Rankings for Expert Designer . . . . . 43



## LIST OF FIGURES

3.1	Consolidated Approach Methodology Diagram . . . . .	11
3.2	Preprocessing Steps . . . . .	14
3.3	Adapter Pattern Example. . . . .	14
3.4	Adapter Pattern After Normalization and Tokenization. . . . .	15
3.5	Adapter Pattern After Stop Word Removal. . . . .	16
3.6	Adapter Pattern After Stemming. . . . .	17
3.7	Term Document Matrix for Intent of Adapter Pattern . . . . .	18
3.8	Term Document Matrix for Intents of Adapter, Façade, and State Patterns . . . . .	21
3.9	Text Based Recommendation Approach . . . . .	23
3.10	Decision Diagram for Specifying the User's Knowledge Level . . . . .	25
B.1	Text Base Recommendation GUI . . . . .	55
B.2	Classifying Designer's Knowledge Level . . . . .	56
B.3	Pattern Question Screen . . . . .	57
B.4	Final Rankings . . . . .	58

# Chapter 1

## Introduction

Good software design has a set of commonly agreed attributes. Maintainability, reuseability, extensibility, robustness are attributes worth mentioning from this set. Reusability helps us to reduce the work by reusing components in multiple places. But many times it can be difficult and time-consuming to achieve reuseability. The way to achieve highest level of reuseability is generalizing the solution. Maintenance implies changes made to the software. Software is an evolving entity, hence, maintainability becomes very important over the lifetime of a software product. Software design must employ principles that make it easy to make changes to parts of the software without breaking its functionality. Extensibility is a closely related attribute to maintainability. In addition to making changes to the already present functionality, new requirements necessitates new extensions to be added to the existing functions. Therefore, software must be designed and implemented with extensibility in mind. While creating a reusable, easily maintainable, and extensible system, one should not compromise robustness. The design should be resilient to changes and new extensions, and in the case of failures, it must correct the failure, or fail gracefully leaving enough evidence indicating why it failed.

Design patterns enhance software design and development process, especially in the areas of maintenance and code reuse. They help the designer choose design alternatives that make a system reusable and avoid alternatives that compromise reusability, thus making the system more maintainable. Design patterns are commonly defined as general, reusable, and time-tested solutions to recurring design problems within a given context. Despite the fact that object oriented paradigm is the focus of design patterns, the philosophy behind design patterns is applicable to non object-oriented systems as well. Design patterns facilitate the development of loosely-coupled and highly-cohesive modules, and make the overall system easier to understand and maintain. Design patterns records the experiences of the previous software designers. This body of knowledge benefits the designer, making readily available a proven solution to a common problem which is applied immediately without the need for rediscovering the solution again. Patterns help a designer get a design “right” more quickly.

Design patterns capture the previous expertise and make it available to designers. But it can be difficult even for the experienced designers to choose the right design pattern for a particular problem. The task of finding the correct pattern can be overwhelming for inexperienced and novice designers. A practical tool that aids the inexperienced designer by recommending a design pattern is very beneficial.

There is research on design pattern recommendation, and tools that implement various methods. Much of these research focused on just one recommendation method. In this thesis, our goal is to provide a combined methodology that will improve the pattern recommendation process. For this purpose, we developed a recommendation tool that combines three different methods that already exist in the literature. These methods are:

1. Text Based Approach

## 2. Case Based Approach

## 3. Question Based Approach

Text-based recommendation approach uses design pattern definitions from different sources such as design pattern books and real world design problems and creates a pattern definition collection. The design problem is then compared against this collection. The tool starts its process by applying the text retrieval techniques to model the document collection and the design problem using vector space model. Once the documents are modeled as vectors, the tool then uses the cosine similarity metric to compare the design problem against the pattern definition collection set, and ranks the recommended design patterns according to this metric. In the second step, existing cases for which a correct answer is already known is used to improve the ranking. Finally a number of pattern questions are asked to the designer to further improve the results. The answers are adjusted based on the expertise level of the designer, which is determined before pattern questions are asked. The tool presents the user a final ranked list of patterns that can be potentially applied to the problem. The tool also provides an opportunity to add newly resolved case scenarios into its repertoire of known cases.

The thesis is organized as follows: Chapter 1 explains the motivation behind the work. Chapter 2 discusses other design pattern recommendation systems that exists in the literature and the methods they use. Chapter 3 presents in depth the methodology used in the thesis and demonstrates the methodology with an example. Chapter 4 discusses the implementation details of the tool. Finally, Chapter 5 gives a discussion of the results. The Appendix lists the sample scenarios used in the thesis and screenshots of the Graphical User Interface.

# Chapter 2

## Literature Review

Design patterns became very popular with the publication of, now enormously famous, Gang-of-Four (GoF) book [1]. It is a very mature topic right now with numerous books and textbooks written on it [2, 3, 4]. When it comes to design patterns, a recommendation system to support inexperienced users in their decision-making when faced with complex design problems is very beneficial. In this section, we shall discuss some of the recommendation systems, approaches, tools and methodologies proposed in the literature. Currently, it is almost inevitable that all recommendation systems use the 23 design patterns from the GoF book. We will investigate recommendation methodologies under 5 categories.

1. Text Based Recommendation
2. Case Based Recommendation
3. Question Based Recommendation
4. Combination of the previous three
5. Miscellaneous Approaches

## 2.1 Text Based Recommendation

Text Based Recommendation systems suggest patterns using a similarity score between the design pattern description and design problem, using text classification and text retrieval techniques developed for natural language processing [5, 6]. The study in [23] uses the popular weighting scheme TF-IDF and machine learning algorithms to recommend a pattern. Three design patterns books are used for training classifiers. 26 real world case studies are used to evaluate the results.

Another study that uses text retrieval in design pattern recommendation is [8]. This study uses the pattern definitions from the Gof book and design problems to create a document collection. After the preprocessing steps, the pattern definitions and design problems are represented using Vector Space Model, and rankings are calculated according to cosine similarity scores. The same authors suggests a slightly different technique in [9]. They use a probabilistic model in design pattern selection. The method suggested in [10] extract the most important words from pattern definitions, and associates these words with a specific pattern, and using them in subsequent comparisons.

## 2.2 Case Based Recommendation

Case Based Recommendation Systems use the previous cases, applications and design problems with known answers, and use this information to suggest a pattern to the current design problem under discussion.

The approach given in [11] is a case based design pattern recommendation system which is built on the previously stored experiences. These experiences are called Design Pattern Application cases and stored in a library. These cases are indexed using method and participant mappings, as such they enable search and



retrieval of an existing case that best matches to the current design problem, and a rank is generated accordingly. This approach is implemented in a CASE tool.

[12] proposes a combination of Case Based Reasoning approach and Formal Concept Analysis techniques. Similar to [11] it uses an indexing mechanism generated from the phrases of the intent sections of the patterns. It uses a similarity metric to retrieve the most similar case, when given a design problem.

## 2.3 Question Based Recommendation

Question Based Recommendation system in general asks a carefully selected design pattern question from a repository then assigns a score based on the answers and recommends a pattern to the designer. Selecting the right question and obtaining accuracy with minimal number of questions is a challenge in those systems. Another consideration in assigning a score is to take into account the knowledge level of the user.

The work in [13] proposed a recommendation system called Design Pattern Recommender with the Goal Question Metric approach, which consists of four steps. The first step forms the conditions from the Intent part of pattern, and in second step forms subconditions from the applicability part of pattern from the GOF book. A tree representation is constructed from these conditions and subconditions, and then questions are derived from this tree representation. In the last step, it constructs Goal - Question - Metric (GQM) where pattern names are goals. Each question is assigned a weight and at the end of the process, positive and negative answers translated into a score and the system recommends a pattern to users.

The study described in [14] is a question/answer based prototype expert system, which recognizes ten design patterns selected from the GoF book. The authors created a set of questions for novice developers and recommend a pattern based on the answers. They experimented the prototype on four students which is rather a low number. The study does not present any quantitative results.

The study given in [15] represents design pattern knowledge using ontology and incorporates an interactive question/answer session to reason about an adequate pattern for a particular design problem. A new ontology called Design Pattern Advisement Ontology is proposed. The pattern knowledge modeled in this ontology is stored in a repository which guides the question/answer session with the developer. The tool has a success rate of 58 percent in identifying the correct design pattern.

The framework presented in [16] creates pattern related questions to help a developer to find a suitable pattern. The framework presents a set of questions based on the properties of the design patterns. Using the developer's answers, the system decides which pattern is suitable to a particular problem. To validate the proposed approach they used a survey that consists of 13 questions, to evaluate the results in terms of relation between design patterns and the problem, experience level of developer and effectiveness of design patterns.

The study in [17], retrieves and asks questions from a repository, matching the intent section of patterns described in the GOF book. They use *pattern\_weighted\_score* measures to give a score based on the answers. The scores are further improved after asking more questions, this time using structure of the pattern.

## 2.4 Combined Approaches

The methodology proposed in [18] has two steps. In the first step the system learns from previous cases, and patterns and in the second step the system interacts with the developers via a questionnaire. The three different techniques - TF-IDF, Latent Semantic Analysis and Principal Component Analysis are employed to find similar cases from other scenarios that are stored previously. According to their results, PCA showed the most promise.

The work presented [19] in consists of two phases. The first phase calculates a similarity between a pattern and a problem by using method names, class names, participants etc. as the criteria. This phase requires that the designer must draw and formulate the design problem in such a way that a similarity matrix can be created. In the second phase, system asks the user questions, and refines the list of design patterns accordingly.

## 2.5 Miscellaneous Approaches

The study in [20] presents a rule-based design pattern recommendation approach implemented in an interactive tool. The system gives the user some semantic criteria and recommendation rules and expects the user to create the problem using them, structurally. As a result of the explanation made by these rules, the system recommends a design pattern to the user.

In the [21], the design problems of inexperienced developers are compared with experienced developers' solutions to previous design problems, and a cosine similarity score is generated. Using this score a design pattern is recommended to the user. The approach given in [22] is an UML based approach. However, unlike other UML- based design patterns recommendation approaches, it proposes a

suggestion that includes not only a UML description of design pattern, but also the UML model and textual information that includes pattern's responsibilities, purposes, advantages etc.

In [?], the design pattern usage descriptions are divided into hierarchical structures, and tabulated. This hierarchical structure and one design problem example is coded as a web application. Experiments were done with the students using the web application. The average accuracy of the application is 74.81%.

One can find dynamic design pattern recommendation approach based on anti-pattern detection in [24], and a design pattern recommendation tool based on multi agent system technology in [25] with agents assigned different tasks of the design pattern recommendation process.

Before we conclude this section we note that researchers use their own scenarios in evaluation their design pattern recommendation systems that they develop. Best to our knowledge, there does not exist a benchmark in the literature. In majority of these studies the scenarios are not explicitly stated. This presents an opportunity to the research community to create a benchmark to evaluate the design pattern recommendation systems.

# Chapter 3

## Methodology

Design pattern recommendation approach used in this thesis is a consolidated one which combines the following three existing techniques from the information retrieval arena to accurately recommend a design pattern for a given design problem scenario:

1. Text Based Recommendation
2. Case Based Recommendation
3. Question Based Recommendation

Several techniques are developed in the informational retrieval arena to find material (usually documents) of an unstructured nature (usually text) that satisfies an information need from a document set. The text based recommendation approach used in the thesis attempts to find a match between the design pattern problem scenario and a design pattern description from within large collection of design pattern descriptions. We shall use the vector space model, n-gram model, term weighting and cosine similarity from the information retrieval domain in

finding a matching design pattern. When text based recommendation process is unable to recommend a pattern in a convincing manner, case based recommendation phase will take over. Case based recommendation is similar to text based recommendation in which the set of design pattern descriptions is augmented with scenarios for which the answer is known. The problem scenario is then checked against this new augmented set. Finally, in the question based recommendation phase a series of pre-determined questions are asked to the designer and a pattern is recommended based on the answer. The answer is adjusted according to the knowledge level of the designer.

The diagram 3.1 below illustrates the flow of the consolidated approach. The process begins with the Data Collection and Assembly step, continues with the Text Base Recommendation phase, followed by merging of the results with the ones obtained in the Question Base Recommendation phase.

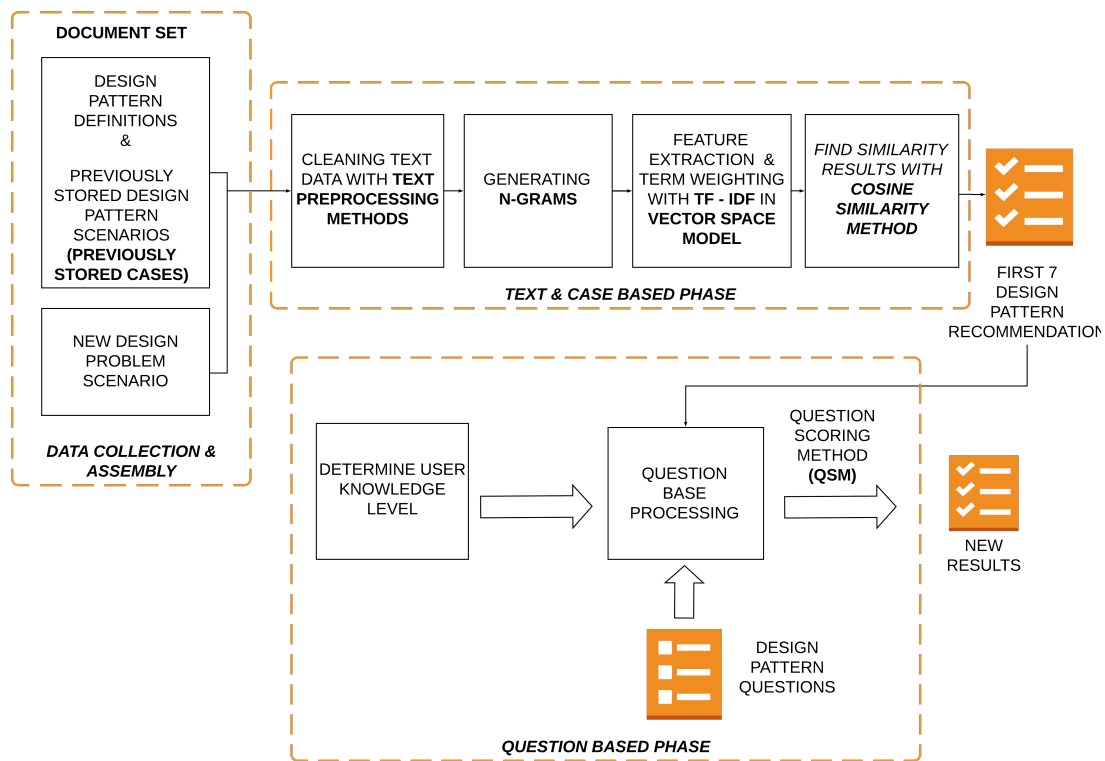


Figure 3.1: Consolidated Approach Methodology Diagram

In the following sections, we will explain in detail the techniques and steps used in this thesis that constitutes our methodology.

## 3.1 Text Based Recommendation

The text based recommendation approach involves four major steps.

1. Design pattern data collection and assembly
2. Text preprocessing
3. Feature extraction and term weighting
4. Similarity computation

### 3.1.1 Data Collection and Assembly

The first step of the text based recommendation process is the collection of the documents containing the design pattern definitions. The pattern definition sources are GoF design pattern book, Wikipedia and a selection of websites that discuss about design patterns problems and solutions [26, 27]. These will constitute our document set used in subsequent steps.

The GoF design pattern book catalogs patterns under several sections, namely, *Intent*, *Motivation*, *Applicability*, *Structure*, *Participants*, *Collaborations*, *Consequences* and *Implementation*. The *Intent*, *Motivation*, *Applicability* and *Participants* sections alone are used when creating design pattern description collection set. We choose these sections, because they contain discriminatory information and keywords when comparing patterns. In addition to Gof design pattern descriptions, we used the overview parts of Wikipedia and some other websites on

design patterns problems and solutions [26, 27] in building our collection set. In our opinion, the language used in Wikipedia pattern descriptions are much closer to the language used in problem scenarios. Consequently, similarity results improved after the addition of Wikipedia text to the collection set. Finally, a list of keywords considered stronger hints for the use of a particular pattern were compiled, and these keywords were added as pseudo-documents to the collection.

The design problem scenarios used in validating the methodology are compiled from the term projects and homeworks of the design pattern course taught at Izmir University of Economics, Software Engineering department. These scenarios had been given to the students over the years and they were asked to find the most appropriate design pattern for the problem. We also compiled several scenarios from the sources referred in [28, 29, 30, 31, 32, 33, 34, 35, 36]. These scenarios were used in our tool to measure the effectiveness of the approach proposed in the thesis.

The distribution of 120 scenarios from the scenario set and their correct pattern names are listed in Table 3.1. The scenarios that we have used in our study covered only 20 patterns. Therefore, our study is conducted for 20 patterns out of 23 and does not include Proxy, Interpreter and Flyweight design patterns.

<b>PATTERN NAME</b>	<b># of SCENARIOS</b>	<b>PATTERN NAME</b>	<b># of SCENARIOS</b>
Observer	15	Composite	12
Abstract Factory	9	Template Method	9
Adapter	8	Visitor	8
Command	7	Singleton	7
Bridge	5	Builder	5
State	5	Strategy	5
Chain of Resp.	4	Facade	4
Iterator	4	Memento	4
Decorator	3	Factory	3
Prototype	2	Mediator	1

Table 3.1: Number of Scenarios for Each Software Design Pattern



### 3.1.2 Text Preprocessing

The data collection and assembly step creates a large document set to be used in selecting the most suitable design pattern. However, this data needs to go through linguistic preprocessing and be prepared for further processing. This preprocessing, shown in Figure 3.2, is a standard technique used in information retrieval and can be further divided into three sub steps:

1. Normalization and Tokenization
2. Stop - Word Removal
3. Stemming

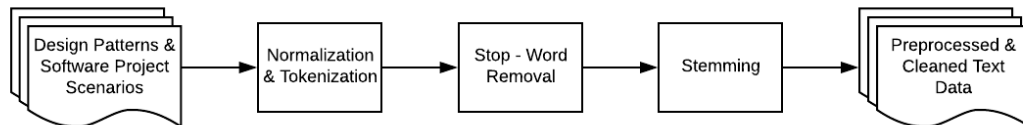


Figure 3.2: Preprocessing Steps

We will demonstrate the preprocessing steps through the Adapter Pattern's intent section which is given in Figure 3.3.

#### **Adapter Pattern Example - Intent**

Convert the interface of a class into another interface clients expect.

Figure 3.3: Adapter Pattern Example.

### 3.1.3 Tokenization and Normalization

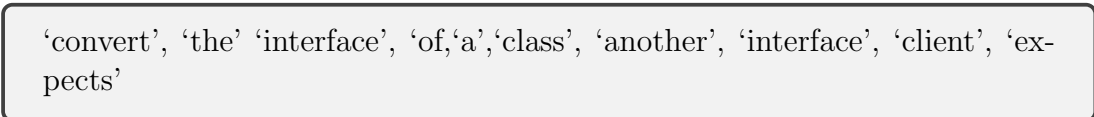
Documents consist of character sequences which are possibly encoded in some binary representation for electronic storage and retrieval. Programming languages

have extensive built-in support to decode this binary representation and group these sequence of characters into what is called a token. Tokenization is the task of generating this group of characters stripped of certain characters such as punctuation, for further syntactic or semantic processing. Despite the fact that *word* and *token* have different meanings we shall use these terms interchangeably in the context of the thesis.

Once the document is separated into its tokens, the normalization phase transforms the text so that it can be processed in a uniform manner. For example the documents may contain numbers and digits. While numbers and digits might help to find some patterns among the documents, most of the times they don't really help to identify the documents. So these will be removed and only alphabetical characters are kept. The normalization process includes tasks such as

- capitalization, case-folding
- removing numbers, converting numbers to their word equivalents
- removing punctuation
- removing white spaces
- expanding abbreviations, expanding contractions
- removing accents and diacritics

and so on. The transformation of the Adapter Pattern after the application of Tokenization and Normalization step is shown in Figure 3.4



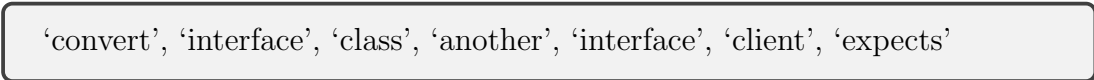
```
'convert', 'the', 'interface', 'of', 'a', 'class', 'another', 'interface', 'client', 'expects'
```

Figure 3.4: Adapter Pattern After Normalization and Tokenization.

## Stop - Word Removal

Words that are so generic would not really help to differentiate one document from another. In this sense they are deemed unimportant because they can be found frequently in every document. These words are called stop words, which are discarded from the document representation. Stop words are typically function words: determiners (a, the), prepositions (on, above), conjunctions (and, but). The stop words that are corpus-specific such as the word “money” in finance corpus will not be a differentiator so they can be removed. However, we make no corpus specific distinction in this thesis. The goal behind removing stop words is that we would like to work with a set of words that are unique to each document. The same set of stop words is used in processing all the documents in the collection

Adapter Pattern definition after the Stop-Word removal is shown Figure 3.5.



```
'convert', 'interface', 'class', 'another', 'interface', 'client', 'expects'
```

Figure 3.5: Adapter Pattern After Stop Word Removal.

## Stemming

The words of a language is inflected with a prefix, suffix or infix to express different grammatical categories such as tense, case, voice, aspect, person, number, gender, and mood. When words are derived from one another, we would like to find out the root form of the word. Stemming is the process of reducing inflection in words to their root forms. A stemmer conflates different variations of a word by reducing them to a common root/stem which may result in words that are not actual words. Adapter Pattern, after the stemming step is shown Figure 3.6.



'convert', 'interfac', 'class', 'anoth', 'interfac', 'client', 'expect'

Figure 3.6: Adapter Pattern After Stemming.

### 3.1.4 Feature Extraction, Term Weighting and Vector Space Model

After the text is pre-processed, we have a new set of document set. Information retrieval and language processing needs a computational model to do reasoning such as similarity and relatedness on the content of documents. The current best model to efficiently access, manipulate and reason about documents is vector space model. Vector Space Model (VSM) is an algebraic model where the documents are modeled as vectors as such there is an axis for every word (term) in the vocabulary. A document is transformed into a vector using the vocabulary of the language. The dimension of the vector representing a document is equal to the number of words in the vocabulary. This means that if the vector space is  $n$ -dimensional, the size of the vocabulary is  $n$ .

Instead of using the the full vocabulary of the English language, in practice one operates on a corpus of fixed number of documents. The vocabulary is established by going over these set of documents and extracting only the unique unigram words (aka terms) contained in those documents to create an unordered list of words. This set of unique words is called bag-of-words because only the terms themselves are included in the bag without any accompanying meta information. The bag-of-words for the document set constitutes the vocabulary. Table 3.2 is an example of bag of words created from intent sections of Adapter, Facade and State patterns.

```

bag-of-words = [ 'convert', 'interfac', 'class', 'anoth', 'interfac', 'client', 'ex-
pect', 'adapt', 'let', 'class', 'work', 'togeth', 'otherwis', 'incompat', 'interfac',
'provid', 'unifi', 'interfac', 'set', 'interfac', 'subsystem', 'allow', 'object', 'alter',
'behavior', 'intern', 'state', 'chang', 'object', 'appear', 'chang', 'class']

```

Table 3.2: Bag of Words

Given the bag-of-words that are extracted from the documents, a feature vector for a given document is created where vector elements may be a binary indicator that tells the existence or absence of the word, or frequency counts or some other kind of ‘weight’ indicating its importance in a document. The number of the dimensions in the feature vector is equal to the number of unique terms in the entire set of documents. VSM is a powerful way of expressing and comparing documents with their weights and values. Despite the fact that the vector representation loses information about the ordering of the words in the document such that one cannot reconstruct the original sentence from the vector, for many tasks the vector approximation is sufficient to capture the meaning of the original document. Figure 3.7 shows the feature vector for the Adapter patterns where weights are defined as frequency counts.

	convert	anoth	client	expect	adapt	let	work	togeth	otherwis	incompat	interfac	class
<b>Adapter Pattern</b>	1	1	1	1	1	1	1	1	1	1	1	1

	provid	unifi	set	subsystem	object	chang	allow	alter	behaviour	intern	state	appear
<b>Adapter Pattern</b>	0	0	0	0	0	0	0	0	0	0	0	0

Figure 3.7: Term Document Matrix for Intent of Adapter Pattern

The document vectors form a term-document matrix (TDM) where rows represents documents, columns represent terms which are typically word types. TDM

is a useful way of modeling the document set. Each cell in the matrix indicates the weight of the term in the set of documents. Typically, the value is a function of the frequency with which the term occurs in the document or in the document collection as a whole.

### 3.1.5 n-gram Features

n-gram model is a *sequence* of “n” tokens that are used in text mining and natural language processing tasks (NLP). It presents the co - occurrence of words in the documents. n is the value at which the repeat is checked. n - gram is a combination of n number words. An 1-gram is referred to as a “unigram” and a 2-gram is referred as a “bigram” which is a a combination of two words. A “trigram” is a combination of three words. Words in the n -gram model, do not necessarily have to be related to each other; they can only be words that follow each other. n - gram model is effective to obtain important statistical information for some text retrieval datasets.

Name of n - Gram	n - gram	Example
<b>Unigram</b>	1 - gram	['class', 'clients', 'convert', 'expect', 'interface']
<b>Bigram</b>	2 - gram	['class interface', 'clients expect', 'convert interface', 'interface class', 'interface clients']
<b>Trigram</b>	3 - gram	['class interface clients', 'convert interface class', 'interface class interface', 'interface clients expect']
<b>Unigram and Bi-gram</b>	1 - gram and 2 - gram	['class', 'class interface', 'clients', 'clients expect', 'convert', 'convert interface', 'expect', 'interface', 'interface class', 'interface clients']

Table 3.3: Extracting n-grams for the Adapter Pattern

In this study, unigram, bigram, and unigram and bigram are used together to try to increase the chance of finding the correct pattern. Examples of unigram, bigram, trigram and unigram and bigram are shown in Table 3.3.

### 3.1.6 TF-IDF

TF-IDF is a weighting mechanism that calculates the importance of each term for each document by increasing the importance based on the term frequency while decreasing the importance based on the document frequency. Terms that are limited to a few documents are useful for discriminating those documents from the rest of the collection; terms that occur frequently across the entire collection are not as helpful.

How many times a given term appears in the document it belongs to is the TF (Term Frequency) part of TF-IDF. It gives the frequency of the word in the document. The higher the TF value is, the more important the term is for the document. The raw frequency is reduced by a logarithmic factor, since a word appearing  $x$  times in a document does not make that word  $x$  times more likely to be relevant to the meaning of the document. However, if the given term appears too frequent —ubiquitous, in all the documents then it is not that important in order to identify the document. For example, if every single document contains ‘the’ ‘chain’ ‘bit’ then these terms would not be helpful to identify any given document. The ‘IDF (Inverse Document Frequency)’ factor of the ‘TF-IDF’ captures this observation by decreasing the importance of a given term as the number of the documents it shows up increases. The formula used in calculating TF-IDF values is given below [6, 37].

*tf\_idf = term frequency \* inverse document frequency*

$$tf\_idf(t, d) = tf(t, d) * idf(t) \tag{3.1}$$

$$idf(t) = \log[n/df(t)] + 1 \tag{3.2}$$

$$tf\_idf(t, d) = tf(t, d) * \log[n/df(t)] + 1 \tag{3.3}$$

where

$\mathbf{n}$  = total number of documents in the document set,

$\mathbf{df}(\mathbf{t})$  = the number of documents in the document set that contain the term  $\mathbf{t}$ .

Each document is represented as a vector whose direction is determined on a set of the TF-IDF values in the space. The vector space model which uses TF-IDF is also called TF-IDF model where each dimension of the vector is weighted using the TF-IDF [6, 37]. An example of the Term Document Matrix using TF-IDF values for Adapter, Façade, and State patterns are given in Figure 3.8.

	convert	anoth	client	expect	adapt	let	work	togeth	otherwis	incompat	interfac	class
Adapter Pattern	0.02703	0.02703	0.02703	0.02703	0.02703	0.02703	0.02703	0.02703	0.02703	0.02703	0.0	0.0
Façade Pattern	-	-	-	-	-	-	-	-	-	-	0.0	-
State Pattern	-	-	-	-	-	-	-	-	-	-	-	0.0

	provid	unifi	set	subsystem	object	chang	allow	alter	behaviour	intern	state	appear
Adapter Pattern	-	-	-	-	-	-	-	-	-	-	-	-
Façade Pattern	0.06758	0.06758	0.06758	0.06758	-	-	-	-	-	-	-	-
State Pattern	-	-	-	-	0.07372	0.07372	0.03686	0.03686	0.03686	0.03686	0.03686	0.03686

Figure 3.8: Term Document Matrix for Intents of Adapter, Façade, and State Patterns

### 3.1.7 Cosine Similarity

After transforming and preparing the text data and giving the scores to each term by calculating the TF-IDF, now one can decide if two documents are similar or not using cosine similarity. It is by far the most common similarity metric which is based the cosine measure of the angle between the vectors. The formula for calculating Cosine Similarity is given in Equation 3.1 [6].



$$\cos(\vec{v}, \vec{w}) = \frac{\vec{v} \cdot \vec{w}}{\|\vec{v}\| \|\vec{w}\|} = \frac{\sum_{i=1}^n v_i w_i}{\sqrt{\sum_{i=1}^n (v_i)^2} \sqrt{\sum_{i=1}^n (w_i)^2}} \quad (3.4)$$

Computing the cosine similarities between the problem scenario vector and each pattern definition vector in the collection, then sorting the resulting scores and selecting the top K documents from the list is the primary idea behind the Text Based Recommendation phase of the methodology. Design pattern recommendation based on text or information retrieval works as follows and shown in Figure 3.9.

1. Treat the problem scenario as a document
2. Calculate the similarity between the problem scenario document and each document in the collection
3. Rank documents by increasing similarity using cosine similarity metric
4. Return to the designer the top k-ranked documents

## 3.2 Case Based Recommendation

Preprocessing of the document set ensures that there are no insignificant words in the document that may adversely affect the design pattern selection. However, the text based recommendation still may be unable to suggest a suitable pattern using the document set constructed from the previously-listed sources. In order to improve the accuracy of the recommendation, Case Based Recommendation is added to the Text Based Recommendation step. Case Based Recommendation simply enriches the document set by augmenting the document set with case scenarios for which the answer is already known. This is because if the problem scenario shows some high degree similarity with a previous known case scenario,

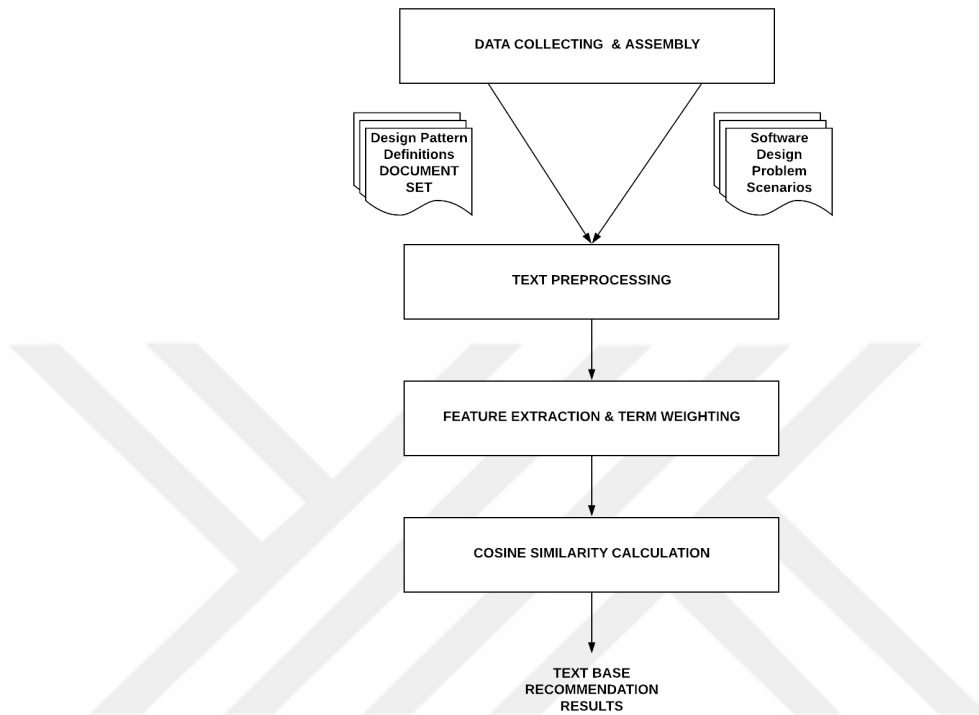


Figure 3.9: Text Based Recommendation Approach

then the problem scenario probably uses the same design pattern. When a problem scenario is marked as such that it uses a specific design pattern, then this problem scenario is designated as a case scenario and can be added to the case scenario set. This will strengthen the prediction capability of the tool as the tool in some way starts “learning” from its previous findings.

### 3.3 Question Based Recommendation

Text Based Recommendation automatically generates result without human intervention. There are specific expressions in the scenarios which are best interpreted by humans. Human involvement in the form of a question / answer session is therefore very beneficial for predicting the correct pattern. This is the approach taken in the Question Based Recommendation phase.

The process is as follows: calculate another score for the Question Based Recommendation phase and adjust the Text Based Recommendation's scores using these new scores. This gives a new and improved ranking which indirectly includes the designer's involvement. Our studies show that the most accurate pattern estimation is within the first 7 results of Text Based Recommendation. In the Question Based Recommendation phase, we focus on these 7 patterns and present questions on these patterns first. However, we present the designer the full set of questions giving the designer the option of answering the remaining questions.

There are two aspects of question/answer session: to determine the knowledge level of the designer, and to gather responses given to pattern questions. Three categories are used to assign a knowledge level to a designer: *Novice*, *Intermediate*, and *Expert*. Determination of the knowledge level of the designer is done by again presenting a series of question given in Table 3.4. An alternative to this could have been to ask the designer to assign a level him/herself. However, this would not have been an objective assessment and the results would have been biased. We have used a subset of the questions given in [38] and added some more.

Questions	Answer Options
Q1: Do you write and publish your design patterns?	a) No b) Yes
Q2: How many design patterns do you know?	a) None (1 pt) b) Less than 23 GoF patterns (2 pts) c) All of the 23 GoF patterns and others (3 pts)
Q3: How many design patterns did you use in the your projects?	a) None (1 pt) b) Less than 23 GoF patterns (2 pts) c) All of the 23 GoF patterns and others (3 pts)

Table 3.4: Questions for Determining Knowledge Level

After having determined the designer's knowledge level, the design pattern questions are presented to the designer. One question is asked for each design

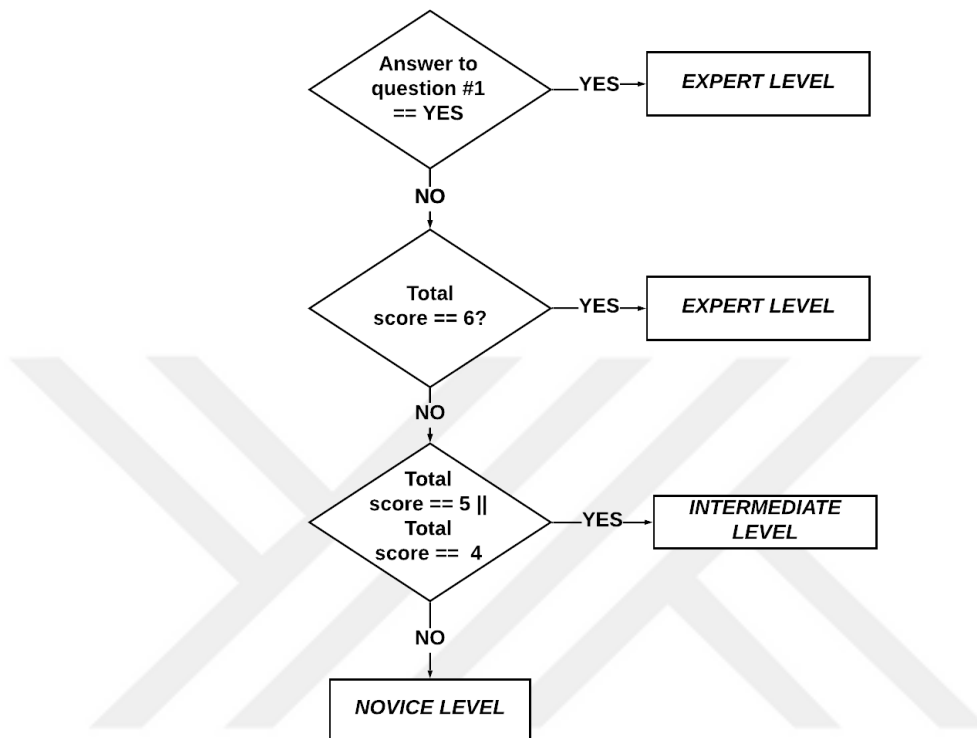


Figure 3.10: Decision Diagram for Specifying the User's Knowledge Level

pattern which makes a total of 20 questions. The patterns selected from the GoF design patterns book are Abstract Factory, Adapter, Bridge, Builder, Chain, Responsibility, Command, Composer, Façade, Factory, Iterator, Mediator, Memento, Observer, Prototype, Singleton, State, Strategy, Template Method and Visitor patterns. Questions were prepared to reflect the characteristic of the pattern and point out the most important feature of the design pattern. The reason that a single question is asked for each pattern instead of several ones is to improve the score without confusing the designer with multiple questions. The questions were prepared by synthesizing and interpreting the information obtained from various sources, especially GoF. We only show a sample of design pattern evaluation questions in Table 3.5 to provide an example. The designer is asked pattern questions corresponding to the first 7 patterns. However, the designer is given the option of answering the remaining questions in the case that the designer believes the rankings are inadequate.

Pattern	Question	1=totally disagree 5=totally agree				
		1	2	3	4	5
Factory	You need to create an object to represent external data or process an external event where another object is responsible for creating and determining the type of this object.					
Singleton	You need to create an instance from a class and provide only one access point to it so all other instances or objects of other classes can access it through the same solely entry point.					
Composite	You need to create an object or objects that inherit an interface and composed from other objects that inherit the same interface.					
Facade	You have a subsystem (could be a package) composed of set of classes or objects and you need to provide a single way to communicate with them through a simpler interface.					
Decorator	You need to create an object that needs different functionalities and responsibilities to be added to and withdrawn from it dynamically at run time.					
Iterator	You have an array or a list or any type of such aggregation structure that you need to access it sequentially.					
Observer	You have objects that need to monitor or observe the changes in the state of each other and need to be noticed whenever the state of any of them has been changed.					
Visitor	You have a complex structure (a hierarchy class structure or an array or list) that contains different elements and need to apply the same different functions, methods, behaviors on these elements.					
Strategy	You have a family of different related classes or algorithms that perform almost the same task into different manners and you need to use them interchangeably.					
Command	You have the situation where there is an action(s) needed to be issued or performed several times, so it is needed to be stored and recalled when needed later on.					
Prototype	You need to create a prototypical instance that is independent of how its products are created, composed, and represented and you need to create new objects by copying this prototype.					

Table 3.5: Pattern Questions

The answers to pattern questions are rated using a five-point numbered Likert scale ranging from 1 to 5 points, where 1 means totally disagree, 2 means disagree, 3 means neutral, 4 means agree and 5 means totally agree. The knowledge level of the designer determined before, is factored in to the answers given to pattern questions, and a final weight is calculated. In this way, designers from different levels of knowledge were prevented from affecting the calculation at the same rate. The knowledge level ranges from 3 to 1, based on the answers given to placement questions as follows: Novice = 3, Intermediate = 2 and Expert = 1. The knowledge level points and points obtained from pattern questions are combined to adjust the scores of the Text Based Recommendation phase. The scores obtained from pattern questions are reduced proportional to the experience level. For example a 5 score from the pattern question will be divided into 3 (i.e.  $5/3 = 1.3$ ) for a novice designer.

The Text Based Recommendation phase scores are now ready to be “de-noised” using the scores obtained from the Question Based Recommendation phase. We have named this “de-noising” operation as “Question Scoring Method (QSM)”. Similar to the approaches used in image processing, we will “de-noise” the results of Text Based Recommendation (i.e. cosine similarity) using the results of Question Based Recommendation. We developed the following formula inspired by [39], to compute a weight that will be applied to the cosine similarity score of the corresponding pattern. This will change the ranking of the patterns because the information obtained from the designer (Expert, Intermediate or Novice), has an impact on the recommendation of the correct pattern. The formula is given below:

$$w(s) = cs_{max} * \frac{e^s}{e^5} \quad (3.5)$$

where  $cs_{max}$  refers the maximum value of the pattern similarity scores obtained in the Text Based Recommendation phase, and  $s$  refers to the value  $\frac{PatternQuestionPoints}{UserKnowledgeLevels}$ . This value can be between  $\frac{1}{3}$  and 5. A novice developer can give minimum 1 point to any question and this means the developer totally disagrees with the statement. Since a novice developer's knowledge level point is 3, the minimum value of  $\frac{1}{3}$  is obtained. An expert developer can give a maximum 5 points to any question meaning that the developer totally agrees with the statement. The knowledge level point is 1 for an expert developer, therefore, the maximum value of 5 is obtained. The weight calculated from QSM is now added to the cosine similarity scores. After the "de-nosing", a new rank is created. This concludes the Question Based Recommendation and also terminates the whole recommendation process.

### 3.3.1 Design Pattern Methodology Example

In this section we shall show how our methodology is applied to a particular design problem. Our pattern collection set is constructed using the GoF design pattern book, Wikipedia and some websites that discusses design patterns problems and solutions [26]. We shall use the following scenario as an example design problem in our demonstration of the methodology.

**Scenario #24:** "Once every five years the library goes through the Science books and records its usage. For the Technology books (Class T) it calculates relevance factor (the average and median age of the books and the ratio of the books that have been published in the past 5 years to the total number of books.)"

The correct answer for this scenario is Visitor Pattern. We shall apply our methodology and demonstrate how it reaches to the correct answer. Table 3.6

shows the results after the Text Based Recommendation phase.

Rank	Pattern Name	Similarity Scores
1	Strategy	0.0203
2	Template Method	0.0159
3	Abstract Factory	0.01473
4	Observer	0.0087
5	Singleton	0.00858
6	Adapter	0.00774
7	Decorator	0.00759
8	<b>Visitor</b>	<b>0.00711</b>
•	•	•
•	•	•
•	•	•

Table 3.6: Text Based Recommendation Results for Scenario #24

As it can be seen from Table 3.6 Visitor pattern is listed 8<sup>th</sup> in the list. Now we enter into Case Based Recommendation phase and add those scenarios as cases to the collection set for which the Text Based Recommendation phase ranks the correct answer as first. The new rankings are shown in Table 3.7.

Rank	Pattern Name	Similarity Scores
1	Facade	0.04359
2	Observer	0.03216
3	Template Method	0.02074
4	Strategy	0.01864
5	Abstract Factory	0.01375
6	<b>Visitor</b>	<b>0.01325</b>
7	Singleton	0.01146
•	•	•
•	•	•
•	•	•

Table 3.7: Text and Case Based Recommendation Results for Scenario #24

The Visitor Pattern is now ranked 6<sup>th</sup> which indicates an improvement. The rankings of other patterns have also changed. The next step is the Question Based



Recommendation phase. First the designer’s knowledge level is determined using the answers to the placement questions listed in Table 3.8. Accordingly the level of expertise for this designer is determined as *Expert*.

Question	Answer	Point
Q1	No	0
Q2	All 23 GoF and more	3
Q3	All 23 GoF and more	3

Table 3.8: Specifying Knowledge Level

Next the designer is asked to answer pattern questions for the 20 patterns. The designer is required to give an answer for questions corresponding to the top 7 patterns listed in the latest ranking. The remaining questions are optional. Knowledge level and scores from these answers are used to calculate a weight using QSM. The calculated weights are shown in Table 3.9.

Question	Answer	Weight
Facade	1	0.0008
Observer	3	0.0059
Template M.	2	0.00217
Strategy	2	0.00217
Abstract F.	1	0.0008
<b>Visitor</b>	5	0.04359
Singleton	3	0.0059
•	•	•
•	•	•
•	•	•

Table 3.9: QSM Results for Scenario #24 for the Expert Designer

These weights are used to obtain the adjusted scores of Text and Case Based Recommendation Phases as shown in Table 3.7. The final rankings are determined which are shown in Table 3.10.

Rank	Pattern Name	FinalRankings
1	<b>Visitor</b>	<b>0.05684</b>
2	Facade	0.04439
3	Observer	0.03806
•	•	•
•	•	•
•	•	•

Table 3.10: Text, Case and Question Based Recommendation Results for Scenario #24

The progress of the ranking improvements are summarized in Table 3.11.

Phase	Rank
Text Based	<b>Eighth order</b>
Text and Case Based	<b>Sixth order</b>
Text, Case and Question Based	<b>First Order</b>

Table 3.11: Summary of Progress in Rankings

# Chapter 4

## Implementation and Results

Design pattern recommendation tool is implemented using Python programming language. The main reason we choose Python for implementation is that it has a robust library support that we need in our processes for natural language processing. We used PyCharm as programming editor for building the user interface for our tool. PyCharm is a free, non-commercial and effective cross - platform IDE, suited well for Python development.

The pattern definition collection set is formed using the GoF book [1], Wikipedia, a website that discusses about design patterns problems and solutions [26] together with the pseudo documents we created from keywords. The use of Wikipedia and other sources in addition to Gof book improved results, because these sources use a language that is much closer to the everyday language used in problem scenarios.

The following scenario demonstrates our justification why we included Wikipedia in the preparation of the collection. Scenario #1 given below is the design problem to which a recommendation is sought after.

**Scenario #1:** “A menu consists of a set of choices and a mechanism for a user to specify which choice they want. There are a variety of styles of menus. One menu style is to print a number in front of each string (e.g., 1, 2, and so on) and let the user enter a number to make a choice. In general, all of the menus must provide a way to add entries to the menu, delete entries, display the menu, and obtain the user’s choice.”

Table 4.1 shows the comparative results when only Gof Book is used versus when Gof and Wikipedia sources are used together.

Rank	GoF		GoF and Wikipedia	
1	Decorator	0.02418	<b>Strategy</b>	<b>0.02623</b>
2	Prototype	0.02184	Factory	0.01867
3	Bridge	0.01855	Mediator	0.01431
4	Observer	0.01447	Prototype	0.01397
5	Factory	0.01342	Facade	0.01275
6	Abstract Factory	0.01122	Composite	0.01207
7	Visitor	0.0106	AbstractFactory	0.01174
8	Facade	0.01018	Bridge	0.01159
9	<b>Strategy</b>	<b>0.01</b>	Visitor	0.01128

Table 4.1: The Effect of Using Wikipedia Sources on Scenario #1

Strategy pattern ranked ninth when the document set uses only GoF, whereas it is ranked first place when the document set contains GoF and Wikipedia together. The use of Wikipedia provided improvements for other problem scenarios not listed here as well. As a result, Wikipedia is included as a source in building the pattern definition collection set.

For text preprocessing step Scikit learn library [37] is used. Scikit learn library is one of the most widely used library for data mining and data analysis. In this implementation, we used this library for text processing, n - gram extraction, tf - idf weighting. For the stemming process, Porter’s Algorithm is used. Porter’s Algorithm is the most popular algorithm for stemming, which has been around since 1979 and its effectiveness has been proven in many other studies.

Unigram and bigram model is used together for n-gram extraction. The trigram experiments did not give any satisfactory results and produced almost 0 value for cosine similarity. For some patterns, it was enough to group them as unigram for a correct recommendation. Some other patterns did not have enough unigram, so grouping them as bigram gave better results. Therefore, it was observed that using unigram and bigram together offers the best combination. For example, when we grouped the document set as unigram, the words “only” and “one” were too frequent to have enough effect for the Singleton pattern; when we grouped these two together as unigram and bigram, the comparison was made with “only”, “one” and “only one” word groups, and as a result, the system was able to recommend the Singleton pattern as correct.

The effectiveness of the methodology was measured using 120 scenarios. Table 4.2 displays the success ratio of the Text Based Recommendation. The correct answer is in the first place for 56 out of 120 scenarios, in second place for 16 scenarios, and third place for 8. This means that 65% of the times the correct answer was listed in the top three, 76% of the times the correct answer was within the top five, and 86% of the times the answer was placed in top 7.

<b>Rank of Correct Pattern</b>	<b># of Scenarios</b>	<b>%</b>
1 <sup>st</sup>	56 scenarios	% 46,66
2 <sup>nd</sup>	16 scenarios	% 13,33
3 <sup>rd</sup>	8 scenarios	% 6,66
4 <sup>th</sup>	7 scenarios	% 5,83
5 <sup>th</sup>	5 scenarios	% 4,16
6 <sup>th</sup>	6 scenarios	% 5
7 <sup>th</sup>	5 scenarios	% 4,16
8 <sup>th</sup> to last	13 scenarios	% 10,83
No recommendation	4 scenarios	% 3,33

Table 4.2: Text Based Recommendation Results

Table 4.3 shows the table representation of confusion matrix of the text-based

recommendation results. In this table, numbers from 1 to 20 refer to the design patterns that we used in our study and number 21 refers to scenarios that do not have similarity scores. This matrix shows how many of the patterns we know to be, for example Abstract Factory as the answer are recommended as Abstract Factory, and how many of them are recommended as Adapter, Bridge, Builder etc. When this table is examined, it provides information about how to improve the document sets for future studies.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
<b>1 (Abstract F.)</b>	4	0	0	0	0	1	0	0	0	1	0	0	1	0	2	0	0	0	0	0	0
<b>2 (Adapter)</b>	2	2	0	0	0	1	0	0	1	0	0	1	0	1	0	0	0	0	0	0	0
<b>3 (Bridge)</b>	0	1	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
<b>4 (Builder)</b>	0	0	0	4	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
<b>5 (Chain of Res.)</b>	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
<b>6 (Command)</b>	0	0	0	0	1	3	0	0	0	0	0	0	0	0	1	0	0	0	0	2	0
<b>7 (Composite)</b>	3	0	0	2	0	1	1	0	1	0	0	0	0	0	1	2	0	0	0	0	1
<b>8 (Decorator)</b>	0	1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
<b>9 (Facade)</b>	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	1	0	0	0
<b>10 (Factory)</b>	2	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
<b>11 (Iterator)</b>	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	0	0	0	0	1
<b>12 (Mediator)</b>	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
<b>13 (Memento)</b>	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	2	0	0	0	0
<b>14 (Observer)</b>	0	0	0	0	0	0	0	0	0	1	0	0	0	11	0	0	0	2	0	1	0
<b>15 (Prototype)</b>	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
<b>16 (Singleton)</b>	2	0	0	0	0	1	0	0	0	0	0	0	0	0	0	3	0	0	0	0	1
<b>17 (State)</b>	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0	0	0	0
<b>18 (Strategy)</b>	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	1
<b>19 (Template M.)</b>	0	0	0	2	0	1	0	0	0	0	0	0	0	0	0	0	1	0	4	1	0
<b>20 (Visitor)</b>	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	3	1	3	0

Table 4.3: Text Based Recommendation Results Confusion Matrix

After the text-based recommendation phase, 56 patterns with the correct answer in the first rank were added to the document set and the text-base recommendation step was repeated. The results of the Case Based Recommendation phase are shown in the table 4.4. As a result of this phase, 4 of the 13 scenarios whose answers were in the 8th to 20th has moved to within the first 7. Two of these scenarios have reached the 5<sup>th</sup> rank and two have reached the 6<sup>th</sup> rank.

Rank of Correct Pattern	# of Scenarios	%
1 <sup>st</sup>	56 scenarios	% 46,66
2 <sup>nd</sup>	16 scenarios	% 13,33
3 <sup>rd</sup>	8 scenarios	% 6,66
4 <sup>th</sup>	7 scenarios	% 5,83
5 <sup>th</sup>	7 scenarios	% 5,83
6 <sup>th</sup>	8 scenarios	% 6,66
7 <sup>th</sup>	5 scenarios	% 4,16
8 <sup>th</sup> to last	9 scenarios	% 7,5
No recommendation	4 scenarios	% 3,33

Table 4.4: Text and Case Based Recommendation Results

Table 4.5 shows the rankings for scenario #18 as an example of accurate recommendation. The correct answer for scenario #18 is Observer pattern and the tool listed Observer pattern as the first pattern in the recommended list at the end of Text Based Recommendation phase with a similarity score of 0.04521.

**Scenario #18:** “Applications use devices of the operating system. Devices generate interrupts for the applications. When the interrupt is handled, the registered applications will be notified and they will take action. For instance, if a character arrives to the network port, the applications waiting for data will be notified and one of them will consume this data.”

Rank	Recommended Pattern	Similarity Scores
1	<b>Observer</b>	<b>0.04521</b>
2	Singleton	0.03922
3	Strategy	0.03301
4	Visitor	0.02419
5	Prototype	0.02208
6	Chain Of Resp	0.01539
7	Command	0.0137

Table 4.5: Scenario #18 Results

14 percent of the cases the correct pattern was not listed in top 7. For those scenarios in the bottom 14%, when we applied the Case Based Recommendation,

results are improved slightly. The correctly identified scenarios from Text Base Recommendation phase are added to the case set. When this new document collection with the added cases are used, 4 of the scenarios previously in the bottom 14% of the list moves to top 7. For the remaining scenarios the designer can perform Question Based Recommendation phase to improve the scores. The designer is required to answer the first 7 pattern questions, For the remaining ones only the questions with an answer are used.

Three such example scenarios given below, - scenario #36, #7 and #20- fall in the bottom 14% percentile. The rankings for these scenarios are shown in Tables [4.6](#), [4.7](#) and [4.8](#).



**Scenario #36:** “You decided to design a Document Management System. A document management system is used to track and store electronic documents. These documents usually belong to a workorder. A workorder is created by the users of the system. A workorder can contain sub workorders. Every document belongs to some workorder. For instance a Purchase work order has two sub workorders: Shipment and Payment. Shipment may have two documents in it: a bill of lading and packaging slip, while Payment may have an invoice document. When time comes you must perform a check that all the documents in the workorder have been signed.”

Rank	Recommended Pattern	Similarity Scores
1	Singleton	0.0146
2	AbstractFactory	0.01128
3	Factory	0.01107
4	Prototype	0.01095
5	Command	0.00983
6	Adapter	0.00967
7	Strategy	0.0093
8	Iterator	0.00869
9	Decorator	0.00859
10	Builder	0.00785
11	Visitor	0.0072
12	<b>Composite</b>	<b>0.0072</b>

Table 4.6: Scenario #36 Results

**Scenario #7:** “A company manufactures several products which are listed in a Product Catalog. A product may consist of other products. Each product has an item number. Customers give orders for these products. An order may have multiple items of different kind or multiple items of same kind. In other words, Orders are composed of Line Items. So you need to keep quantity in each Line Item. An Order could be canceled or committed. An item can be cancelled until it is committed. Based on the contents of the order, the customer will be charged accordingly.”

Rank	Recommended Pattern	Similarity Scores
1	AbstractFactory	0.05629
2	Builder	0.0514
3	Factory	0.03276
4	Prototype	0.02398
5	Facade	0.00801
6	Observer	0.00735
7	Bridge	0.00629
8	Memento	0.00609
9	Iterator	0.00508
10	Strategy	0.00382
11	Decorator	0.00372
12	<b>Composite</b>	<b>0.00337</b>

Table 4.7: Scenario #7 Results

**Scenario #20:** “In a library the books are organized using Library of Congress Classification Outline. That is Class Q is for Science class QA is for Mathematics and Class QA75.5-76.95 is for Computer Science (i.e. SCIENCE->MATHEMATICS->COMPUTER SCIENCE). You can even further classify Computer Science as Database or AI.”

Rank	Recommended Pattern	Similarity Scores
1	Singleton	0.02709
2	Adapter	0.02217
3	Strategy	0.02003
4	Visitor	0.01882
5	Decorator	0.01867
6	Factory	0.01826
7	Bridge	0.01798
8	AbstractFactory	0.01725
9	Prototype	0.01636
10	Mediator	0.0145
11	State	0.01256
12	TemplateMethod	0.0112
13	Facade	0.01011
14	Builder	0.00741
15	<b>Composite</b>	<b>0.00716</b>

Table 4.8: Scenario #20 Results

The correct answer for these three design problems, scenarios #36, #7 and #20, is Composite Pattern. When we examine the above scenarios and their results, scenarios #36 and #7 are listed at the 12<sup>th</sup> place among the 20 patterns and scenario #20 is listed at the 15<sup>th</sup> place. These three scenarios are in the bottom 14% percentile which is indication that text based recommendation did not produce any satisfactory result. Text Based Recommendation automatically generates results without any human intervention. But as it is seen from these three scenarios #36, #7 and #20, there are specific expressions in them which are best interpreted by humans. Human involvement in the form of a question/answer session is very beneficial for predicting the correct pattern. For this reason,

Question Based Recommendation phase is performed to improve the results.

Tables 4.9, 4.10 and 4.11 show the results of QSM and the rankings before and after the application of the QSM.

SCENARIO#36 - NOVICE (=3)							
Rank	Pattern Name	Text Based Results	Pattern Question Response	Norm. Score	QSM Results after de-noise	New Ranking	New List
1	Singleton	0.0146	3	1,00	0.00027	0.01487	Singleton
2	Abstract F.	0.01128	1	0,33	0.00014	0.01142	AbstractF.
3	Factory	0.01107	1	0,33	0.00014	0.01121	Factory
4	Prototype	0.01095	1	0,33	0.00014	0.01109	Prototype
5	Command	0.00983	1	0,33	0.00014	0.00997	Command
6	Adapter	0.00967	1	0,33	0.00014	0.00981	Adapter
7	Strategy	0.0093	1	0,33	0.00014	0.00944	Strategy
8	Iterator	0.00869	1	0,33	0.00014	0.00883	Iterator
9	Decorator	0.00859	2	0,67	0.00019	0.00878	Decorator
10	Builder	0.00785	1	0,33	0.00014	0.00799	Builder
11	Visitor	0.0072	2	0,67	0.00019	<b>0.00772</b>	<b>Composite</b>
12	<b>Composite</b>	<b>0.0072</b>	5	1,67	0.00052	0.00739	Visitor
13	Facade	0.00598	1	0,33	0.00014	0.00612	Facade
14	Bridge	0.00532	1	0,33	0.00014	0.00546	Bridge
15	Template	0.00474	1	0,33	0.00014	0.00488	Template
16	State	0.00454	1	0,33	0.00014	0.00468	State
17	Memento	0.00368	1	0,33	0.00014	0.00382	Memento
18	Observer	0.00315	3	1,00	0.00027	0.00342	Observer
19	ChainOfR.	0.00288	1	0,33	0.00014	0.00302	ChainOfR.
20	Mediator	0.00198	1	0,33	0.00014	0.00212	Mediator

Table 4.9: Scenario #36 Final Rankings for Novice Designer

SCENARIO #7 - INTERMEDIATE (=2)							
Rank	Pattern Name	Text Based Results	Pattern Question Response	Norm. Score	QSM Results after de-noise	New Ranking	New List
1	AbstractF.	0.05629	2	1,00	0.00103	0.05732	AbstractF.
2	Builder	0.0514	2	1,00	0.00103	0.05243	Builder
3	Factory	0.03276	3	1,50	0.0017	0.03446	Factory
4	Prototype	0.02398	1	0,50	0.00063	0.02461	Prototype
5	Facade	0.00801	1	0,50	0.00063	0.00864	Facade
6	Observer	0.00735	1	0,50	0.00063	<b>0.00799</b>	<b>Composite</b>
7	Bridge	0.00629	1	0,50	0.00063	0.00798	Observer
8	Memento	0.00609	1	0,50	0.00063	0.00692	Bridge
9	Iterator	0.00508	1	0,50	0.00063	0.00672	Memento
10	Strategy	0.00382	1	0,50	0.00063	0.00571	Iterator
11	Decorator	0.00372	2	1,00	0.00103	0.00445	Strategy
12	<b>Composite</b>	<b>0.00337</b>	5	2,50	0.00462	0.00475	Decorator
13	Command	0.00323	1	0,50	0.00063	0.00386	Command
14	Visitor	0.00319	1	0,50	0.00063	0.00382	Visitor
15	Template	0.00308	1	0,50	0.00063	0.00371	Template
16	ChainOfR.	0.00294	1	0,50	0.00063	0.00357	ChainOfR.
17	State	0.0021	1	0,50	0.00063	0.00303	Singleton
18	Adapter	0.00162	1	0,50	0.00063	0.00273	State
19	Singleton	0.00133	3	1,50	0.0017	0.00225	Adapter
20	Mediator	0.0009	1	0,50	0.00063	0.00153	Mediator

Table 4.10: Scenario #7 Final Rankings for Intermediate Designer

SCENARIO #20 - EXPERT (=1)							
Rank	Pattern Name	Text Based Results	Pattern Question Response	Norm. Score	QSM Results after de-noise	New Ranking	New List
1	Singleton	0.02709	3	3,00	0.00367	<b>0.03425</b>	<b>Composite</b>
2	Adapter	0.02217	1	1,00	0.0005	0.03076	Singleton
3	Strategy	0.02003	1	1,00	0.0005	0.02267	Adapter
4	Visitor	0.01882	1	1,00	0.0005	0.02053	Strategy
5	Decorator	0.01867	2	2,00	0.00135	0.02002	Decorator
6	Factory	0.01826	1	1,00	0.0005	0.01932	Visitor
7	Bridge	0.01798	1	1,00	0.0005	0.01876	Factory
8	AbstractF.	0.01725	1	1,00	0.0005	0.01848	Bridge
9	Prototype	0.01636	1	1,00	0.0005	0.01775	AbstractF.
10	Mediator	0.0145	1	1,00	0.0005	0.01686	Prototype
11	State	0.01256	1	1,00	0.0005	0.015	Mediator
12	Template	0.0112	1	1,00	0.0005	0.01306	State
13	Facade	0.01011	1	1,00	0.0005	0.0117	Template
14	Builder	0.00741	1	1,00	0.0005	0.01061	Facade
15	<b>Composite</b>	<b>0.00716</b>	5	5,00	0.02709	0.00791	Builder
16	Command	0.00467	1	1,00	0.0005	0.005167	Command
17	ChainOfR.	0.00466	1	1,00	0.0005	0.00516	ChainOfR.
18	Observer	0.00398	1	1,00	0.0005	0.00448	Observer
19	Iterator	0.00134	2	2,00	0.00135	0.00269	Iterator
20	Memento	0.00124	1	1,00	0.0005	0.00174	Memento

Table 4.11: Scenario #20 Final Rankings for Expert Designer

The inclusion of Question Based Recommendation in the methodology has improved the rankings, but less for novice knowledge level when compared to intermediate and experienced knowledge levels. This is expected because novice developers' answers are less reliable than those of experienced designers. The highest improvement is obtained in the experienced level. This is also expected and also desired, as the degree of the knowledge of the designer compensates the shortcoming of the Text Based Recommendation for particular patterns.

When a scenario is "solved" and assigned a designed pattern to it, the problem scenario becomes a known case and added to the document set as described in

the Case Based Recommendation Phase. However, this process is not automatic and left to be done by the designer. In particular the results obtained from novice and intermediate developers are not taken into account while adding a case to the document set. The tool ‘offers’ the problem scenario for inclusion as a case only for the experienced users. And it is just an ‘offer’. It is worthwhile to note that the users of the tool has the full control over what to add or not. The user after receiving a recommendation may further assess the appropriateness of the recommendation and decide to classify it as a case or not. This will help to improve the accuracy of the tool for subsequent recommendations.

# Chapter 5

## Conclusion

In this study, we developed a design pattern recommendation system which combines three different approaches: Text Based Recommendation, Case Based Recommendation and Question Based Recommendation. The goal of the system is to help inexperienced developers to select the most appropriate design pattern for a design problem. This consolidated approach produces better results than using each approach alone. Text Based Recommendation uses a well known technique from Natural Language Processing domain to compare documents. A design problem written in natural language (i.e. English) and stored in a document is compared against a pattern definition collection set using cosine similarity metric. Pattern definition collection set is formed using Gof design pattern book, Wikipedia and web sites that discuss design pattern problems and solutions. 20 GoF design patterns are used. Case Based Recommendation augments the design pattern definition collection with known cases. The design problem is then compared against this augmented set. Finally in the Question Based Recommendation phase, a series of questions asked to the designer to adjust the scores obtained in the previous phases. We developed a Question Scoring Method (QSM) to adjust and improve the rankings obtained in the previous two phases.



The recommendation tool is implemented using Python programming language with a graphical user interface. Natural Language processing libraries of Scikit Learn [37] is used for text preprocessing, and Python is used for the Text Based Recommendation Phase. The effectiveness of the methodology was measured using 120 scenarios. The scenarios are collected from various sources including a design pattern course where these scenarios were actually asked to students in homeworks and projects. At the end of Text and Case Based Recommendation phases, the correct answer is in the first place for 56 out of 120 scenarios, in second place for 16 scenarios, and third place for 8. This means that 65% of the scenarios the correct answer was listed in the top three, in 76% the correct answer was within the top five, and in 86% the answer was placed in top 7. The Question Based Recommendation phase was conducted by software developers with different degrees of experience working for a software company. The answers of the experienced developers has significantly improved the rankings of the incorrectly selected patterns, and in certain cases, the pattern is correctly identified in the first rank. At the end of the Question Based Recommendation phase, for all 9 scenarios, the correct answer was listed in the top 3, with the points given by expert designers. The correct answer was listed in the top 8, with the answers points by intermediate designers. The experimental results showed that a consolidated approach produces better results.

There are a couple areas that the tool needs improvement. Firstly, even though cosine similarity is a well accepted a similarity metric, other similarity measures and weight computations or a combination of them thereof need to be tried. Corpus specific stop word removal can be incorporated to improve results. Secondly, there are keywords that are not present in the design pattern definitions but are good indicators of a particular design patterns. One such example is the words ‘every’ and ‘each’ which are good hints for Iterator pattern. New set of words need to be investigated and added. Thirdly, our Question Scoring Method

can be improved for better de-noising. Lastly the sample problem set needs to grow to include more scenarios to further improve the results.



# Appendix A

## Design Patterns Scenarios

**Scenario #1:** “A menu consists of a set of choices and a mechanism for a user to specify which choice they want. There are a variety of styles of menus. One menu style is to print a number in front of each string (e.g., 1, 2, and so on) and let the user enter a number to make a choice. In general, all of the menus must provide a way to add entries to the menu, delete entries, display the menu, and obtain the user’s choice.”

**Scenario #2:** “The Company class is the central class that encapsulates several important features related to the system as a whole. It is required to make sure that only one instance of this important class can exist.”

**Scenario #3:** “The system has an interface named “MediaPlayer”. This interface is implemented by a concrete class AudioPlayer. AudioPlayer has methods that play mp3 format audio files. There is another interface AdvancedMediaPlayer which is implemented by a concrete class AdvancedAudioPlayer to play vlc and mp4 format files. It is required to have AudioPlayer class to use AdvancedAudioPlayer class to be able to play other formats.”

**Scenario #4:** “The designer of an adventure game wants a player to be able to take and drop various items found in the rooms of the game. Two of the items found in the game are bags and boxes. Both bags and boxes can contain individual items as well as other bags and boxes. Bags and boxes can be opened and closed and items can be added to or taken from a bag or box.”

**Scenario #5:** “The system approves purchasing requests. There are four approval authority. The selection of the approval authority depends on the purchase amount. If the amount of the purchase is higher than 1 million dollar, the owner who approves. If it ranges from 500k to less than 1 million the CEO who approves, if it ranges from 25k to less than 500k the head of department approves, if less than 25k the vice who approves. The approval authority for a given dollar amount could change at any time and the system should be flexible enough to handle this situation.”

**Scenario #6:** “The system should have only one printer spooler although the system can identify many printers.”

**Scenario #7:** “A company manufactures several products which are listed in a Product Catalog. A product may consist of other products. Each product has an item number. Customers give orders for these products. An order may have multiple items of different kind or multiple items of same kind. In other words, Orders are composed of Line Items. So you need to keep quantity in each Line Item. An Order could be canceled or committed. An item can be cancelled until it is committed. Based on the contents of the order, the customer will be charged accordingly.”

**Scenario #8:** “The order application system is grouped into three subsystems: Product subsystem, Order subsystem, Customer subsystem. The Customer subsystem keeps information about the customer. The classes in one subsystem may have the information about the classes in another subsystem. For instance, Line Item Class in the Order Subsystem must know about Product class in the Product subsystem.”

**Scenario #9:** “We have D1, D2 and D3 devices. Each device’s reset behavior is different. Whenever we reset a device we must log a generic deviceReset event which does not change from device to device. In addition to having different reset behaviors, our devices have different reset interfaces. For example, for device D1 a reset means shutdown followed by a reboot, for device D2 reset means suspend followed by reboot and for device D3 it means reboot. We would like to uniformly reset these devices.”

**Scenario #10:** “The operating system has D1, D2 and D3 devices. These devices are organized into subsystems. Subsystems may also contain other subsystems. We want to reset the whole system by pushing a reset button.”

**Scenario #11:** “Company X has a generic assembly plant which can build car for different brands when given parts of a car. Together with the parts, an assembly manual that gives detail steps of the assembly process is also provided to the assembly plant. This assembly plant is careful enough not mix the parts of one brand with another.”

**Scenario #12:** “In the same city, there two manufacturing plants which make engine, chassis, steering and transmission for two different models of the same company. These parts then shipped to the assembly plant for the final assembly. These parts are composed of sub parts and each part has an associated cost with it. When the assembly process is completed the price of the car is stuck to the windshield for potential buyers.”

**Scenario #13:** “Engine is assembled by the engine team; chassis is assembled by the chassis team and so forth. Chassis is assembled first followed by engine and followed by transmission. Finally steering is put in place. This order must be preserved. After the assembly plant gets the parts and manual, it waits until the headquarters give the order to build the car.”

**Scenario #14:** “A farmer owns a farm open to public for picking your own vegetables for a fee. The farm is organized as a collection of rows where each row has a different vegetable planted. For example, row one has cucumbers, row two has green beans and row three has green pepper. Upon entry to the farm each customer is given a basket that has compartments for each vegetable to ease sorting of the vegetables. In our example the basket would have three compartments.”

**Scenario #15:** “At each end of the row there is a sensor and a camera that monitors the rows (catching people eating the vegetables). Whenever a person enters or exits a row the sensor activates the camera and the image gets transferred to a central monitoring room.”

**Scenario #16:** “The farmer plants winter vegetables for winter which have different picking process than summer vegetables (i.e. using hand versus a tool).”

**Scenario #17:** “The Operating System has a single files syetm in which files are organized in a hierarchical manner. Folders contains other folders and files. Some folders are encrypted some are not. One cannot create an unencrypted file in an encrypted directory.”

**Scenario #18:** “Applications use devices of the operating system. Devices generate interrupts for the applications. When the interrupt is handled, the registered applications will be notified and they will take action. For instance, if a character arrives to the network port, the applications waiting for data will be notified and one of them will consume this data.”

**Scenario #19:** “The OS will be using CPU, hard disk, and I/O devices. Before shutting down the OS each device must be properly “reset”. Each device’s reset behavior is different. Whenever we reset a device we must log a generic deviceReset event which does not change from device to device. In addition to having different reset behaviors, our devices have different reset interfaces. For example, for hard disk device a reset means first write whatever data in the buffer then close the files. For CPU it means terminate all processes. We would like to uniformly reset these devices. We want to reset the OS by issuing a shutdown command. Note that a system has fixed number of CPU’s, hard disks and other devices.”

**Scenario #20:** “In a library the books are organized using Library of Congress Classification Outline. That is Class Q is for Science class QA is for Mathematics and Class QA75.5-76.95 is for Computer Science (i.e. SCIENCE->MATHEMATICS->COMPUTER SCIENCE). You can even further classify Computer Science as Database or AI.”



# Appendix B

## Recommendation Tool GUI

The graphical user interface is created using the PyQt Designer. The user interface design is then converted to a structure that can be understood by the PyCharm Python editor. In the following the screen shots of the tool are presented. The designer, after invoking the design pattern recommendation tool, is shown the screen depicted in Figure [B.1](#). The designer enters the problem scenario, a score is calculated using the Text Based Recommendation approach. A list of recommended design patterns are displayed.

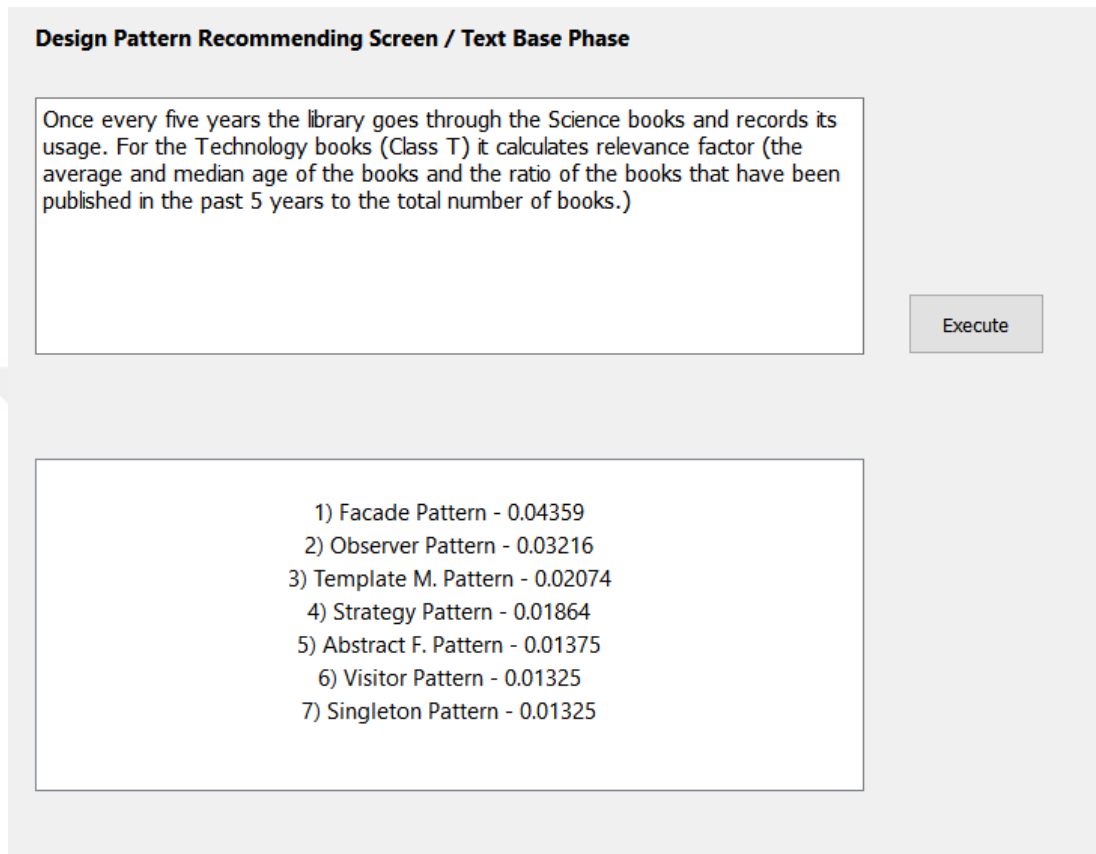


Figure B.1: Text Base Recommendation GUI

The following screen shown in Figure B.2 is used to ask the placements questions in order to determine the knowledge level of the designer. One of the levels; Novice, Intermediate or Expert is assigned to the designer based on the answers. The calculation method has been explained in Figure 3.10.

**Design Pattern Knowledge Level Classifying**

Q1: Do you write and publish your design patterns?

No

Yes

Q2: How many design patterns do you know?

None

Less than 23 GoF patterns

All of the 23 GoF patterns and others

Q3: How many design patterns did you use in the your projects?

None

Less than 23 GoF patterns

All of the 23 GoF patterns and others

Figure B.2: Classifying Designer's Knowledge Level

## APPENDIX B. RECOMMENDATION TOOL GUI

Once every five years the library goes through the Science books and records its usage. For the Technology books (Class T) it calculates relevance factor (the average and median age of the books and the ratio of the books that have been published in the past 5 years to the total number of books.)

Your Level  
**EXPERT**

1) You have a subsystem (could be a package) composed of set of classes or objects and you need to provide a single way to communicate with them through a simpler interface.  
 1       2       3       4       5

2) You have objects that need to monitor or observe the changes in the state of each other and need to be noticed whenever the state of any of them has been changed.  
 1       2       3       4       5

3) You need to create subclasses that redefine certain steps of your algorithm without changing the algorithm's structure.  
 1       2       3       4       5

4) You have a family of different related classes or algorithms that perform almost the same task into different manners and you need to use them interchangeably.  
 1       2       3       4       5

5) You need to create an object to represent external data or process an external event where another object is responsible for creating and determining the type of this object.  
 1       2       3       4       5

6) You have a complex structure (a hierarchy class structure or an array or list) that contains different elements and need to apply the same different functions, methods, behaviors on these elements.  
 1       2       3       4       5

7) You need to create an instance from a class and provide only one access point to it so all other instances or objects of other classes can access it through the same solely entry point.  
 1       2       3       4       5

8) You need to create an object or objects that inherit an interface and composed from other objects that inherit the same interface.  
 1       2       3       4       5

9) You need to create an object that needs different functionalities and responsibilities to be added to and withdrawn from it dynamically at run time.  
 1       2       3       4       5

10) You have an array or a list or any type of such aggregation structure that you need to access it sequentially.  
 1       2       3       4       5

11) You have the situation where there is an action(s) needed to be issued or performed several times, so it is needed to be stored and recalled when needed later on.  
 1       2       3       4       5

12) You need to create a prototypical instance that is independent of how its products are created, composed, and represented and you need to create new objects by copying this prototype.  
 1       2       3       4       5

13) You need to convert the interface of a class into another interface clients expect because of incompatible interfaces.  
 1       2       3       4       5

14) You need to just one construction process that can create different representation.  
 1       2       3       4       5

15) You want to share an implementation among multiple objects with the decoupling an abstraction from its implementation.  
 1       2       3       4       5

16) You need to chain the receiving objects and pass the request along the chain until an object handles it.  
 1       2       3       4       5

17) You need to define an object that controls how a set of objects interact and vary their interaction independently.  
 1       2       3       4       5

18) You want to capture and externalize an object's internal state without violating encapsulation.  
 1       2       3       4       5

19) You want to allow an object to change its behavior when its internal state changes.  
 1       2       3       4       5

20) You need to create an object to represent external data or process an external event where another object is responsible for creating and determining the type of this object  
 1       2       3       4       5

Figure B.3: Pattern Question Screen

Using the screen shown in Figure B.3, the designer gives a score ranging from 1 to 5 to the questions corresponding to the first 7 recommended pattern. The problem scenario and the knowledge level of the designer is also shown on the same screen. A new rank is calculated and presented using the QSM method. Figure B.4 shows the new rank.

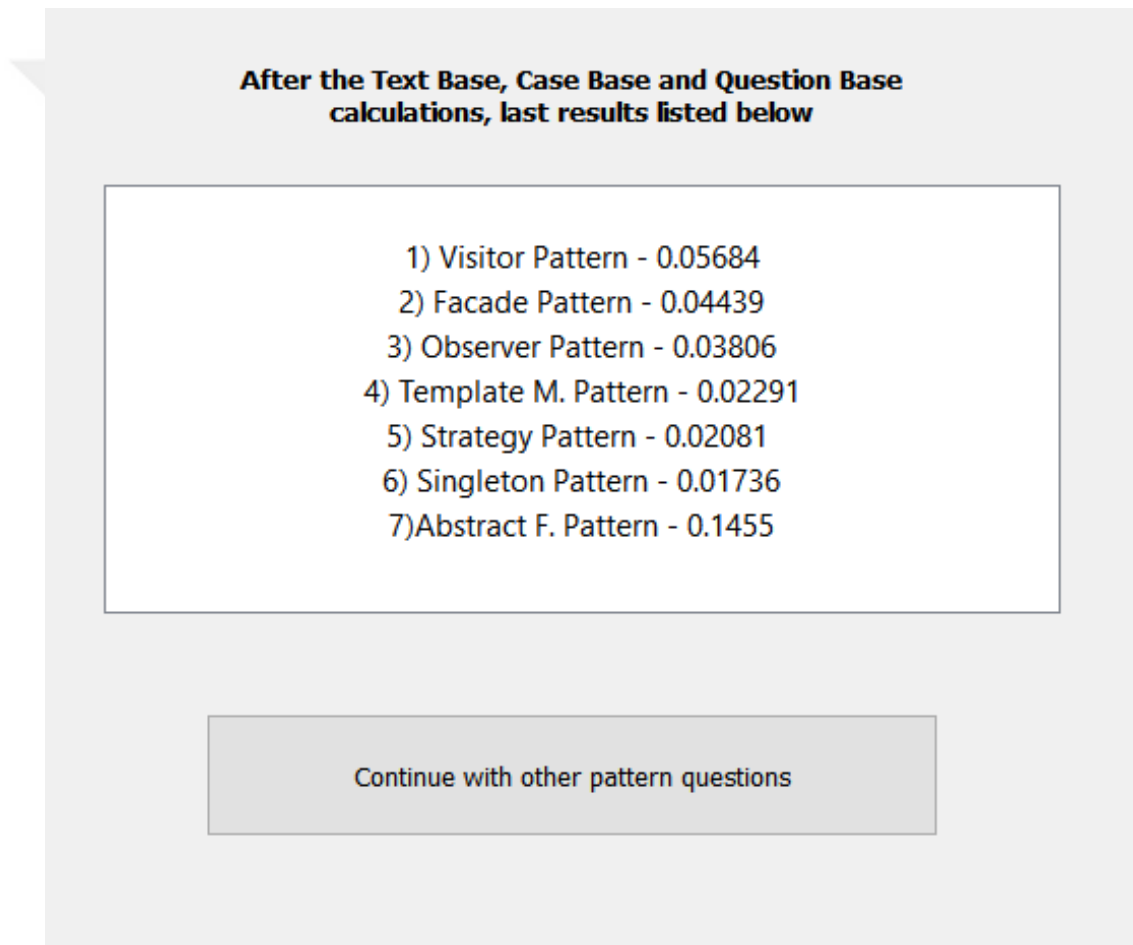


Figure B.4: Final Rankings

The designer now has the option of accepting this new rank or may choose to continue with the remaining pattern questions. The same process is repeated until the designer calls it over.

## BIBLIOGRAPHY

- [1] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). “Design Patterns: Elements of reusable object-oriented software”. Massachusetts: Addison-Wesley Publishing Company.
- [2] Freeman, E., Robson, E., Bates, B. and Sierra, K., (2004). “Head first design patterns”. ” O’Reilly Media, Inc.”.
- [3] Lang, J.E., Bogovich, B.R., Barry, S.C., Durkin, B.G., Katchmar, M.R., Kelly, J.H., McCollum, J.M. and Potts, M., (2001). “Object-oriented programming and design patterns”. ACM SIGCSE Bulletin, 33(4), pp.68-70.
- [4] Shalloway, A., and Trott, J. R. (2005). “Design patterns explained: A new perspective on object-oriented design, 2/E”. Pearson Education India.
- [5] Manning, C., Raghavan, P. and Schütze, H., (2010). “Introduction to information retrieval”. Natural Language Engineering, 16(1), pp.100-103.
- [6] Jurafsky, D. and Martin D.H (2009). “Speech & language processing”. 2nd ed. Prentice Hall, 2009.
- [7] Sanyawong, N. and Nantajeewarawat, E., 2015. “Design pattern recommendation: A text classification approach”. 6th Int. Conf. Inf. and Comm. Tech. for Embedded Systems, IC-ICTES 2015.

- [8] Hamdy, A. and Elsayed, M., (2018). “Automatic Recommendation of Software Design Patterns: Text Retrieval Approach”. *JSW*, 13(4), pp.260-268.
- [9] Hamdy, A. and Elsayed, M., (2018). “Towards More Accurate Automatic Recommendation Of Software Design Patterns”. *Journal of Theoretical & Applied Information Technology*, 96(15).
- [10] Guéhéneuc, Y.G. and Mustapha, R. (2007). “A simple recommender system for design patterns”. *Proceedings of the 1st EuroPLoP Focus Group on Pattern Repositories*.
- [11] Gomes, P., Pereira, F.C., Paiva, P., Seco, N., Carreiro, P., Ferreira, J.L. and Bento, C., (2002). “Using CBR for automation of software design patterns”. In *European Conference on Case-Based Reasoning*. pp. 534-548. Springer, Berlin, Heidelberg.
- [12] Muangon, W. and Intakosum, S., (2013). “Case-Based reasoning for design patterns searching system”. *International Journal of Computer Applications*, 70(26), pp.16-24.
- [13] Palma, F., Farzin, H., Guéhéneuc, Y. G., and Moha, N. (2012). “Recommendation system for design patterns in software development: An dpr overview”. In *2012 Third International Workshop on Recommendation Systems for Software Engineering (RSSE)*. pp. 1-5. IEEE.
- [14] AlSheikSalem, O. and Qattous, H., (2017). “An expert system for design patterns recognition”. *International Journal of Computer Science and Network Security (IJCSNS)*, 17(1), pp. 93-101.
- [15] Pavlič, L., Podgorelec, V., and Heričko, M. (2014). “A question-based design pattern advisement approach”. *Computer Science and Information Systems*, 11(2), pp.645-664.

- [16] Qureshi, M.R.J. and Al-Geshari, W., (2017). “Proposed Automated Framework to Select Suitable Design Pattern”. *International Journal of Modern Education and Computer Science*, 9(5), pp.43-49.
- [17] Suresh, S., Naidu, M., Kiran, S. A., and Tathawade, P. (2011). “Design pattern recommendation system: a methodology, data model and algorithms”. *ICCTAI'2011*.
- [18] Bouassida, N., Jamoussi, S., Msaed, A., and Ben-Abdallah, H. (2015). “An interactive design pattern selection method”. *J. UCS*, 21(13), pp.1746-1766.
- [19] Issaoui, I., Bouassida, N., and Ben-Abdallah, H. (2015). “A new approach for interactive design pattern recommendation”. *Lecture Notes on Software Engineering*, 3(3), pp.173-178.
- [20] Bouassida, N., Kouas, A., and Ben-Abdallah, H. (2011). “A design pattern recommendation approach”. In *2011 IEEE 2nd International Conference on Software Engineering and Service Science*. pp. 590-593. IEEE.
- [21] Birukou, A., Blanzieri, E., and Giorgini, P. (2006). “Choosing the right design pattern: an implicit culture approach”. University of Trento.
- [22] Petri, D. and Csertán, G. (2003). “Design pattern matching”. *Periodica Polytechnica Electrical Engineering*, 47(3-4), pp.205-212.
- [23] Sanyawong, N. and Nantajeewarawat, E. (2014). “Design pattern recommendation based-on a pattern usage hierarchy”. In *2014 International Computer Science and Engineering Conference (ICSEC)*. pp. 134-139 . IEEE.
- [24] Smith, S. and Plante, D. (2012). “Dynamically recommending design patterns”. In *SEKE*. pp. 499-504.



## BIBLIOGRAPHY

---

- [25] Salah, E. M., Zabata, M. T., and Sallabi, O. M. (2013). “Dps: Overview of design pattern selection based on mas technology”. In Distributed Computing and Artificial Intelligence. pp. 243-250. Springer, Cham.
- [26] Sourcemaking Design Patterns Website. URL: <https://sourcemaking.com/design-patterns>, [24 May 2019].
- [27] Refactoring.guru Design Patterns Website. URL: <https://refactoring.guru/design-patterns>, [24 May 2019].
- [28] Bartsch, M. and Harrison, R. (2007). “Design patterns with aspects: A case study”. In EuroPLoP (pp. 797-810).
- [29] Design Pattern Scenario Examples. URL: <https://dzone.com/refcardz/design-patternschapter=1>, [24 May 2019].
- [30] Facade Pattern Scenario Example. URL: <http://techtuts.in/facade-designpattern-with-real-world-example-booking-system/>, [24 May 2019].
- [31] Abstract Factory Pattern Scenario Example. URL: <https://airbrake.io/blog/design-patterns/abstract-factory>, [24 May 2019].
- [32] Design Pattern Scenario Examples. URL: <https://www.javagists.com/introduction-design-patterns>, [24 May 2019].
- [33] Design Pattern Scenario Examples. URL: <https://exceptionnotfound.net/tag/creational-patterns>, [24 May 2019].
- [34] Design Pattern Scenario Examples. URL: <https://www.binpress.com/factory-design-pattern/>, [24 May 2019].
- [35] Design Pattern Scenario Examples. URL: <https://stackify.com/designpatterns-explained-adapter-pattern-with-code-examples/>, [24 May 2019].

## BIBLIOGRAPHY

---

- [36] Factory Design Pattern Scenario Examples. URL: <https://howtodoinjava.com/design-patterns>, [24 May 2019].
- [37] Scikit - Learn Machine Learning in Python. URL: <https://scikitlearn.org/stable/>, [24 May 2019].
- [38] Sahly, E. M. and Sallabi, O. M. (2012). “Design pattern selection: A solution strategy method”. In 2012 International Conference on Computer Systems and Industrial Informatics (pp. 1-6). IEEE.
- [39] Turkan, M. (2011). “Novel texture synthesis methods and their application to image prediction and image inpainting”. PhD thesis, Université Rennes 1, 2011.