SOFTWARE DESIGN PATTERN RECOMMENDATION USING TEXT CLASSIFICATION TECHNIQUES

By

Amanuel Negash Tiruneh

A Thesis Submitted to Department of Computing

School of Electrical Engineering and Computing

Office of Graduate Studies

Adama Science and Technology University

July 2020

Adama, Ethiopia

SOFTWARE DESIGN PATTERN RECOMMENDATION USING TEXT CLASSIFICATION TECHNIQUES

By

Amanuel Negash Tiruneh

Advisor: Bahiru Shifaw (PhD.)

A Thesis Submitted to Department of Computing

School of Electrical Engineering and Computing

Office of Graduate Studies

Adama Science and Technology University

July 2020

Adama, Ethiopia

Declaration

I hereby declare that this MSc thesis is my original work and has not been presented as a partial degree requirement for a degree in any other university and that all sources of materials used for the thesis have been duly acknowledged.

Name: Amanuel Negash

Signature:_____

This MSc thesis has been submitted for examination with my approval as a thesis advisor.

Name: Bahiru Shifaw (PhD.)

Signature:_____

Date of Submission:_____

Approval Page

We, the undersigned, members of the Board of Examiners of the final open defense by "*Amanuel Negash*" have read and evaluated his/her thesis entitled "*Software Design Pattern Recommendation using Text Classification Techniques*" and examined the candidate. This is, therefore, to certify that the thesis has been accepted in partial fulfillment of the requirement of the Degree.

Name of Student	Signature	Date
Supervisor/Advisor	Signature	Date
Chairperson	Signature	Date
Internal Examiner	Signature	Date
KULA KEKEBA (PhD)		31/08/2020
External Examiner	Signature	Date
Head of Department	Signature	Date
School Dean	Signature	Date
Post graduate Dean	Signature	Date

Acknowledgment

First and foremost, I would like to give all the praises to the Almighty God for giving me another day of life to finish this thesis.

I would like to express my sincere gratitude to my advisor ***Bahiru Shifaw (PhD.)*** for the continuous support of my master thesis, and his motivation, patience, and knowledge. Because of his enthusiasm, I value and respect his opinion. His guidance helped me throughout this research and writing of this thesis. I could not have imagined having a better advisor and mentor for my master's study. I appreciate your help. Thank you.

Besides my advisor, I am grateful to ***Mesfin Abebe (PhD.)***, the postgraduate coordinator of the CSE department for his valuable pieces of advice and comments when I choose the thesis topic. I extend gratitude to ***Dr. Shahid Hussain*** from the Department of Computer Science, the University of Hong Kong. He gave me one of a kind comments through email when I asked him about the problems that he intended to address as future work from his published papers, provided me a dataset, and showed me some directions to approach them.

This thesis would not be completed without my graduate friends for their general insights on the thesis. My classmates, especially members of Smart Software SIG also deserve my sincerest thanks, their friendship and assistance have meant more to me than I could ever express. Again, many thanks to ASTU ICT department employees while I was working there, I could not have this best opportunity in my academic life to fulfill my ambition and curiosity.

I give thanks to my dearest mother ***Emebet Kasaye*** for the reason to start this M.Sc. study. Etete, I am always grateful to have a mother like you. I also thank my father, my beloved wife ***Shalom Muluwork***, my handsome son ***Ethan Amanuel***, my supportive brothers ***Dawit Negash*** and ***Henock Kasaye***, whose moral support helped me to complete my M.Sc. thesis. I have no words to express their meticulous love and support.

Table of Contents

Contents

List of Tables

List of Figures

List of Equations

List of Abbreviations

AMI	Adjusted Mutual Information
Arg Max	Argument of the Maxima
BOW	Bag of Words
CBR	Case Base Reasoning
CPU	Central Processing Unit
DBN	Deep Belief Network
DF	Document Frequency
DFS	Distinguishing Feature Selector
DP	Design Pattern
FN	False Negatives
FP	False Positives
GI	Gini Index
GoF	Gang-of-Four
ICA	Independent Component Analysis
IEEE	Institute of Electrical and Electronics Engineers
IG	Information Gain
KNN	K-Nearest Neighbors
MI	Mutual Information
ML	Machine Learning
NB	Naïve Bayes
NLTK	Natural Language Toolkit
NMI	Normalized Mutual Information
OR	Odds Ratio
RF	Random Forest
RI	Random Index
SE	Software Engineering
SMOTE	Synthetic Minority Over-sampling Technique
SOA	Service-Oriented Architecture
SRS	Software Requirement Specification
SVM	Support Vector Machines
TF	Term Frequency
TFC	Term Frequency Collection
TF-IDF	Term Frequency and Inverse Document Frequency
TN	True Negative
TP	True Positive
UML	Unified Modeling Language
VSM	Vector Space Model

Abstract

Software design is considered as a challenging task in the agile software development life cycle, where the fundamental structure of software artifacts is highly provoked to its evolution over time by adding new features or modifying the existing functionality. This affects the system-level quality attributes such as reusability, maintainability, and understandability. Therefore, software design patterns introduce a common practical approach to improve the design quality of the software. The classification scheme and semantic correlation between patterns depend on the experience and knowledge of experts in the corresponding domain. Consequently, a novice developer needs enough knowledge and efforts to understand the classification scheme, the semantic correlation between patterns, and the consequences of each pattern. Text classification-based approaches have been shown in greatness among other software design pattern recommendation methods. However, the inconsistency in text classification schemes and lack of semantically illustrative feature set to organize design patterns are the main constraints to use the existing machine learning models to find a candidate design pattern class and suggest a more appropriate pattern(s). Thus, in this study, we have selected feature extraction methods to exploit an experiment using a text classification-based via supervised learning techniques such as SVM, NB, KNN, and RF. These classifiers are employed to organize similar design patterns by constructing models and recommend the right design pattern group. To recommend the right design pattern, we use cosine similarity measures to compute the degree of closeness between design problems and patterns of suggested pattern class. The models trained using TF-IDF, word2vec, and word2vec weighted by TF-IDF feature extraction methods. To assimilate the importance of the proposed method, the study employed a comparative experiment to determine the best combination of feature extraction methods with their respective machine learning algorithms. A case study is conducted on GoF design pattern collection and along with 38 real software design problem scenarios to evaluate the classification performance. According to the experiment evaluation result the SVM classifier based on Word2Vec weighted by TF-IDF achieves better performance with F1-Score 81.6%, followed by NB with F1-Score of 79.1%. RF model is the least performing model with all feature extraction methods.

Key Words: Software Design Pattern, Software Design Problems, Text Classifications, Supervised Classifiers, Software Design Pattern Recommendation

CHAPTER ONE

1 Introduction

1.1 Background

Software design is one of the software engineering phases that define software methods, functions, objects, and the overall structure and interaction of code so that the resulting functionality will satisfy users' requirements. Software design is considered the most challenging process, as most expansive errors are often introduced during the design phase. Software designers don't immediately come up with a finished design, and instead, they develop design iteratively through several different versions. The beginning point is the informal software design, which is filtered by adding information to make it consistent and complete. Different software design methods existed during the life cycle of software design. For example, the designer uses the common practice of using architecture, components, component design, algorithms, data structures to tackle design problems. These groups of methods use different software architectural styles that define a set of components and connectors that help in coordination, communication, and cooperation between these components. The main benefits of using software architectural styles are the reuse, interoperability, understandability, and vocabulary of design components. A software developer is responsible for selecting software architecture styles based on design methods and applies different architectural strategies and design patterns that fulfill the quality attributes. For example, During the working procedures Attribute-Driven Design (ADD) architecture design method, a software developer selects an architectural style and chooses the desired quality attributes included in the software system. Afterwards, software designers use the description of software design requirements corresponding selected software architectural components and the appointed software design patterns [1]. Software design patterns can be understood as reusable tools that help to solve recurring problems in the software development process. They emerged from the work of Alexander et al. in the field of software architecture [2]. Using software design patterns leads to an increase in software maintainability, reusability, quality, and also reducing the technical risk of the software project by not having to develop and test a new design.

Design patterns may simplify and speed up the software development process as they indicate a direction for a solution that is mostly based on thorough knowledge and experience for a particular issue. Design patterns have become an integral part of many development approaches, and there is a rapid increase in their types. The problem of selecting patterns still exists while the number of documented patterns has continually increased. For instance, Rising's Pattern Almanac[3] lists more than 1200 patterns. Subsequently, Rising - Patterns Almanac 2000 [4], euroPLoP [5], and Portland Pattern Repository [6] are good sources of pattern retrieval from a different domain. The rapid increase in the number of patterns has been reported in the form of documented literature and online repositories, which make it hard to be aware of the design patterns and make the proper selection process as a hand problem [7], supported through the following Sommerville's quote [8] :

> "Only experienced software engineers who have a deep knowledge of patterns can use them effectively. These developers can recognize generic situations where a pattern can be applied. Inexperienced programmers, even if they have read the pattern books, will always find it hard to decide whether they can reuse a pattern or need to develop a special-purpose solution."

The experienced and inexperienced developers follow the same procedure to select and use the design pattern to the description of the design problem [7].

The searching of an appropriate pattern or group of patterns is highly related to a well-defined classification scheme, which is introduced by developer/practitioner experience, and the documentation of frameworks and patterns which might be time-consuming for the developers to yield in the creation of many variants [9]. The existing efforts of developers in the domain of classification of design patterns can be categorized into two groups that are a) Problem-based classification, and b) Solution-based classification schemes. For example, in the domain of object-oriented design problems, Gamma et al. [10], Pree. [11] and Coad et al. [12] classified design patterns into groups for design problems related to specific aspects such as structure, abstract coupling, behavior, aggregation, and interactions of methods and classes. Similarly, in the domain of real-time applications, the design issues to architectural design, memory, safety and reliability, concurrency, resources, and distributions are considered to classify the patterns of real-time applications [13].

The text classification approach has been applied in many domains to automate the systems to decrease the cost and computational time. Subsequently, the text classification approach aid to classify the texts into proper classes to their contents, such as spam e-mail filtering

[14], web page classification [15], sentiment analysis [16], author identification [17], and the design pattern recommendation [15], [18]–[20].

In the domain of design pattern recommendation, text classification techniques have been exploited through supervised learning for the introduction of an automated system. In addition to promising results, there are few limitations to the implementation of the proposed automated system.

Feature vectors extracted from the problem domain of design pattern collection and real design problem scenario descriptions were not considered their semantic illustrative features. This undesirable effect of useless features also affects the performance of supervised classifiers to organize software design patterns into their groups. Also, the lack of well-defined supervised classification schemes with feature extraction methods not investigated experimentally for software design pattern automation using text classification techniques.

The above issues were addressed by this study proposed a method which is based on the Text classification [15], [18]– [20], that aims at employing supervised learning techniques to organize design patterns using selected feature extraction methods by extracting important features for the better classification performance. The classifiers recommend the appropriate design patterns(s) group for a given design problem. Afterward, specific design patterns were recommended by using feature vectors similarity technique. Finally, the text classification classifiers with their corresponding feature extraction methods tested to assess the effectiveness of the proposed method.

1.2 Motivation

Software design is an essential phase of the software development cycle and has an all-pervasive impact on product quality. In solving design problems, design decisions must be performed, whether it is a new problem or a recurring problem. For instance, Gang-of-Four (GoF) design patterns constitute one of the most famous group of design patterns for object-oriented programming. They are divided into three groups, i.e., creational patterns, structural patterns, and behavioral patterns, based on the types of class/object manipulation. Conceptually, creational design patterns deal with object creation mechanisms in different situations; structural design patterns provide suitable ways to organize relationships between entities; and behavioral design patterns identify common communication patterns between objects. Since object creation is an obvious intention, it is relatively easy to determine

whether patterns in the first group should be considered. However, structural relationships and object collaboration patterns are not independent; a given structural pattern usually leads to a certain object interaction pattern, and conversely, a behavioral pattern usually requires a specific structural organization. As a result, when a specific design problem is given, it is often difficult for an inexperienced software designer to determine whether a search for a suitable pattern should be made in the second group or whether it should be made in the third group. Another difficulty arises from a large number of patterns currently available. Even for the experienced programmer, it is challenging for a designer to choose ones without any tool support.

The text classification based automated systems for software design pattern are based on the problem description of patterns instead of class/object manipulation described in the design patterns books. The experimental results of software design pattern recommendation using text classification methods motivated an improvement on feature extraction algorithms to consider more semantically illustrated feature set while training the machine learning models to automate the recommendation process .Therefore, study must be carried to explore effect feature extraction algorithms on the improvement of software design pattern recommendation via text classification techniques. Thus, this will help the developers to describe their design problems in natural language and identify design patterns easily as a solution.

1.3 Statement of the Problem

The employment of design patterns in software development is considered as an alternative approach to software refactoring to address the quality of software. They have advantages to increase the reusability and software modularization. However, from the last two decades, many software design patterns with its spoiled patterns have been introduced and cataloged [9], which creates difficulties for novice to select the right design pattern to solve a design problem without the support of any tool.

Many automated systems have been proposed. The text classification approach is one of them for the recommendation of software design patterns. This method used the textual descriptions of design problem to train their classifier for later recommendation of software design pattern. Moreover, software design problem must deliver the full fledge message and

knowledge of the occurring software design issues. Therefore, the semantic meaning of texts must be captured during the feature extraction construction process.

However, extraction of a semantical illustrative feature set is not considered for training of the classifiers on software design pattern automation using text classification methods [15], [18], [19]. Therefore, this designing feature set for text classifiers on automation of software design pattern is a great deal of mileage.

To mitigate the problems mentioned above, we raised the following research questions.

1. What are the better feature extractions algorithms for software design pattern recommendation using a text classification scheme to employ on supervised learning classifiers?
2. How will the adoption of text classification techniques facilitate the automation of the software design pattern recommendation process?
3. What result will be exhibited in the ensemble of term weighting methods with the pre-trained word embeddings i.e. Word2Vec? How does this affect the performance of the supervised classifiers for recommending a design pattern group and design pattern(s)?

1.4 Objectives

1.4.1 General Objective

- The objective of the research study is to design and develop a text classification method for software design pattern recommendation (selection) from the list of applicable patterns.

1.4.2 Specific Objectives

The specific objectives of this research are:

- To build datasets for training classifiers and testing of the proposed method.
- To identify useful features extraction and supervised machine learning algorithms for software design pattern recommendation.
- To organize software design patterns based on the expert classification scheme via supervised learning techniques.

- To suggest or recommend software design pattern classes/groups for the software design problem.
- To recommend the right design pattern from the suggested pattern class/group using text similarity techniques for a given design problem.

1.5 Scope and Limitations

1.5.1 Scope

Because of a head tight schedule, the research work is limited to on the recommendation of software design patterns by using text classification methods by divulge the effects of TF-IDF, Word2Vec and the combination of these feature extraction techniques on supervised learning algorithms using such as SVM, K-NN, Naïve Bayes, and RF with
their default parameters.

1.5.2 Limitation

During designing and developing of the design pattern recommendation models test dataset i.e. real design problems are needed from software development industries that help to evaluate the proposed method. Unfortunately, there is a lack of companies or industries which documented design problems. As a result, the real design problem scenarios for the proposed research is going to be extracted from the literature reviews and books. Thus, the impact of additional data set and current experimental environment will change the outcome of the result.

Due to time constraints, the proposed method's uses a case study only on the problem description of GoF design pattern collections.

1.6 Beneficiaries and Application of Results

The primary beneficiaries of this study are any stakeholders that are using software design patterns in the software development process. Enough knowledge is required to understand the classification schemes and to find the right patterns. For example, in the case of Gang-of-Four (GoF) (patterns are grouped into 3 categories) and Douglass's (patterns are grouped into five categories) pattern collections, a novice developer can easily understand the nomenclature of classification scheme due to a smaller number of pattern classes. However, in case of Booch [21], interdisciplinary [22], and the spoiled pattern [9], it becomes difficult to understand the classification scheme, and long description of documented patterns [7].

Therefore, this study can be significantly help novice developers to describe their design problems in natural language and allows easily access design patterns.
The other beneficiaries are fellow researchers because they get a chance to replicate and extend this study as well as it could be a base for other related work.

1.7 Organization of the thesis

The thesis work is organized into seven chapters. *Chapter one* introduces the study comprising the background of the thesis, motivation, statement of the problem, research questions, objectives, methodology, the scope of the thesis, limitation as well as the application of the result.

Chapter two of this thesis discusses the literature review and related work part of the research work. In this section, an overview of software design patterns and their applications in the domain of software engineering. It reviews the design pattern classification scheme to group a similar pattern, design pattern selection for the given design problem, the current classification of texts, and its applications on the software design pattern automation.

Chapter three has compiled the methodology of the study used to carry out the thesis work. It confers the overview of the proposed method where software design patterns, the proposed methods such as supervised learning classifiers and feature extraction methods, the data collection, and evaluation methods are clarified. In contrast, *chapter four* explains the proposed method and the techniques used in the research.

Chapter five discusses the implementation and experimentation of the proposed solution. It covers the working environment, dataset description, the implementation of preprocessing, feature extraction implementation, models implementations, and evaluation models. *Chapter six* discusses the result of feature extractions, supervised models, and also presents the major results obtained by comparing the models based on features extraction techniques. *Chapter seven*, the last chapter of the thesis, talks about research conclusion, recommendation, and discovery of future works that would be taken as further research focus from this research topic perspective.

CHAPTER TWO

2 Literature Review

2.1 Overview of Software Design Patterns

A design pattern provides bundled, reusable, tried, and test solutions for such design problems quality and maintainability that required, in addition to reducing the technical risk of the project by not having to develop and test a new design. These patterns provide experiences and knowledge of many designers, and different repositories are compiled for different types of design problems [9].

Software design patterns are considered as an essential way to get the collective wisdom of the software engineering (SE) community and aid to give a tool for practical software engineering. However, the organization and documentation of the collective knowledge of the software engineer community are still in a continual process [24], [25]. The software engineering community is making its efforts to introduce and publish new patterns to resolve the recurring design problems of different domains. In the 1970s, the famous mathematician and architect Alexander introduced the concept of patterns and pattern languages [2]. Alexander has recommended that a pattern should be described in three sections named Context, Description of the problem, and the Solution. Each section can be described in sub-section depending on the developer's interest. However, the template followed to publish the patterns of a domain.

The developers that are the audience of the corresponding domain not only need to understand the functionality of any source code but also require more information to know the robustness and fragility of an implemented design. The well-defined documentation of efforts of developers to resolve the design problems can aid other developers for effective communication. The concept design pattern is adopted and the basis on specific objectives, which include effective communication between developers, software products, and to discuss the declined quality of software. The effective communication between the developers depends on certain features of cataloged patterns such as Clarity, Consistency, Completeness, Conciseness, Concreteness, Courtesy, and Concreteness, which can be affected by several factors such as

1. The interest and experience of the developers who introduce a new pattern catalog, which might be a cause for the introduction of more than one catalogs in the same domain. For example, in object-oriented development [11], [23] have addressed aspects and used different techniques (new or borrowed) to resolve a recurring design problem.

2. The state-of-art developers follow to describe the pattern in a particular section based on their interest either in the same or different domains. For example, PLoP is the best venue to analyze the state-of-the-art, which is followed to catalog the patterns.

Alexander has introduced a template that consists of three sections named Context, Problem description, and a corresponding Solution [2]. Subsequently, the building components of each section are varied across the pattern catalog depending on the developer's experience and knowledge in the response of handling design problem(s).

For instance, in the area of object-oriented software development, Gamma et al. [10] have introduced 23 design patterns, and each pattern classified into two sections named Problem Domain and Solution Domain. The Problem Domain section of each pattern is classified into intent, motivation, and applicability sub-sections. In contrast, the Solution Domain section is classified into the structure, participant collaboration, consequences, implementation, and related patterns. Gamma et al. [10] have also described each pattern with a set of logical sections named intent, motivation, applicability, structure, participants, collaboration, consequences, implementation, sample code, and related patterns. Moreover, in the domain of Service Oriented Architecture (SOA), the authors have described each pattern with a set of 6 sections named Problem, Solution, Application, Implementation, Principles, and architecture. The same concept is adopted by many researchers to introduce the design patterns in different domains. These are object-oriented development [9]–[11], [23], Fault-Tolerant telecommunication [24], communication software [11], web designing [26], Human-Computer Interaction [27], real-time system [13], database management, and network security [28].

Table 2.1 Summary of domains and related frequently used design pattern collections

Patterns source	Sections	Problem Definition Sections	Solution Sections
Gamma et al. [10]	Intent, motivation, applicability, structure, participants, collaboration, consequences, implementation, sample code, and related patterns	Intent, motivation, applicability	Structure, participants, consequences, implementation, sample code, related patterns
Duyne et al. [26]	Patten title, Problem statement, Background information, forces, solutions, related pattern	Patten title, Problem statement, Background information, forces	Solution, Application, Implementation, and architecture
Douglass [13]	Abstract, Problem, Pattern Structure, Collaboration Roles, Consequences, Implementing strategies, Related Patterns	Abstract, Problem	Consequences, Implementing Strategies, Related Patterns
Baraki et al. [29]	Name, Intent, Motivation, forces, and Context, Solution, Consequences, elated Patterns	Name, Intent, Motivation, forces, and Context	Solution, Consequences, Related Patterns

2.2 Software Design Pattern Classification

Several research papers can be used by a developer to recommend the right design pattern(s) to resolve a design problem. These include books, proceedings of conferences/journals, online repositories, and online libraries/groups. From Table 2.1, it can be described that the structure of patterns in different domains and the classification scheme to group a similar pattern is highly related to the developer's knowledge, experience, and interest. For example, Gamma et al. [10], Coad et al. [12], and Pree et al. [11] present their classification scheme to group the design patterns to certain aspects in the domain of object-oriented development. Gamma et al. [10] considers the instantiation, composition, and communication of class objects and introduces the 23 design patterns as a solution for most commonly occurring recurrence design problems, and group them into three broad categories named Creational, Structural, and Behavioral.

Moreover, Coad [12] considers the stereotypical responsibilities and scenario interactions aspects to investigate 31 patterns employed by developers of five well-known industry

applications and group them into four categories named Transaction, Aggregate, Plan, and Interaction. The name of industrial applications is;

- Connie's Convenience Store,
- Wally's and llie's Order Center,
- Dani's Divrters and,
- Andi's Autopilot

Subsequently, W. Pree [8] focuses on the Recursive Structures and Abstract Coupling between objects to classify the pattern collection. The template used to describe the structure of a pattern is related to the domain, developer's knowledge, and experience, and the technique (new or borrowed) to describe the solution. From Table 2.1, we can also extract information that is the domain of design pattern classification where existing efforts of researchers can be summarized as Problem-based classification and solution-based classification

2.2.1 Problem-based Classification of Design Patterns

In the problem-based classification of design patterns, the authors mostly used the context and intent of design problems and applicability of design patterns to organize the patterns into high-level categories. For example, Gammal et al. [10] introduced 23 design patterns and organized them into three top-level categories on the basis of the problem description.

The patterns of the same categories have relevance to each other up to a certain extent. However, it cannot be replaced as the right choice with any different pattern of the same category. Consequently, Problem Definition Domain (Table 2.1) is mostly used by authors to classify the pattern into high-level categories. The components of the Problem Definition Domain section are described in the form of narrative text without the constraint of a borrowed (or new) technique used to describe the solution of a pattern. The problem-based classification schemes, including their domain and high-level categories, are summarized in Table 2.2.1.

Besides the well-defined organization of design patterns with pre-defined classification schemes (shown in Table 2.2.1), there are certain domains where patterns have been introduced without any classification schemes. These are education, the pattern used by developers during the software development life cycle, cooking, embedded systems,

enterprise applications, and automated systems. In the same domain, the existence of several pattern catalogs affects the searching of the right patterns to resolve a design problem. The organization of published design patterns (without any classification scheme) either in the form of books and proceeding, or hosted on an online catalog/library is still required.

Table 2.2 Summary of Problem-based Design Pattern Collections

Author	Domain	Total Patterns	High-Level Categories
Gamma et al. [10]	Object-oriented Design Problems	**23**	Creational, Structural and Behavioral
Pree [11]	Object-oriented Design Problems	**23**	Pree's classification scheme relies on Recursive Structures and Abstract Coupling
Coad et al. [12]	Object-oriented Design Problems	**31**	The transaction, Aggregate, Plan, and Interaction
Tichy [30]	Software Design Problems	**100**	Decoupling, Variant Management, State Handling, Control, Virtual Machines, Convenience, Compound, Concurrency, and Distribution
Rising [3]	Application Type	**1200**	Accounting, Hypermedia, Trading, C++ Idioms, System Modeling, Air Defense, Interactive, Database, Persistence and Trading
Douglass [13]	Real-Time	**34**	Architectural Design, Safety and Reliability, Memory, Concurrency Resources and Distribution
Schumacher [28]	Security Patterns	**46**	SACA (System Access and Control Architecture), ACM (Access Control Model), IA (Identification and Authentication), OSAC (Operating System Access Control), SIA (Security Internet Application), FA (Firewall Architecture), ESRM (Enterprise Security and Risk Management) and Accounting
Duyne et al. [26]	Web Designing	**90**	Site Genres, creating a navigation Framework, creating a Powerful Homepage, Writing and Managing Content, Building Trust and Credibility, Basic E-Commerce, Advanced E-Commerce, Helping Customers complete Tasks, Designing Effective Page Layouts, Making Site Search Fast and relevant, and speeding Up Yout Site
Baraki et al. [29]	Interdisciplinary design patterns	**8**	Classified on the base Socio-technical requirements

2.2.2 Solution-based Classification of Design Patterns

The Software Engineering (SE) research community has made little effort in the domain of solution-based classification of design patterns into the appropriate number of design pattern classes. However, they have used the solution domain of design patterns in designing and implementing specific tools that can aid in identifying or detecting the pattern instances in the given source code. The list of design pattern detection tools in different domains of software engineering has been provided in section 2.3.1. In the domain of object-oriented development, researchers have made their efforts to organize patterns by investigating the solutions of design patterns.

Zimmer [31] proposed an approach to investigate the relationship between 20 Gang-of-Four (GoF) design patterns and organize them accordingly. There are three aims of Zimmer's work;

- To classify the patterns on the base of a relationship exists between them.
- To introduce a new pattern that can be derived from the generalization process of many design patterns, and
- To describe the structure of the design pattern on several layers.

According to the Zimmer approach, the relationship between design patterns can be represented as (X, Y) pairs of design patterns. From Figure 2.1, Zimmer extracts three types of relationships between design patterns such as

- X used Y in its Solution,
- X is similar to Y, and
- X can be combined with Y.

Subsequently, Zimmer extracts re-organize the patterns with three architectural layers, such as application domain, typical software problem, and basic design pattern and techniques, shown in Figure 2.1

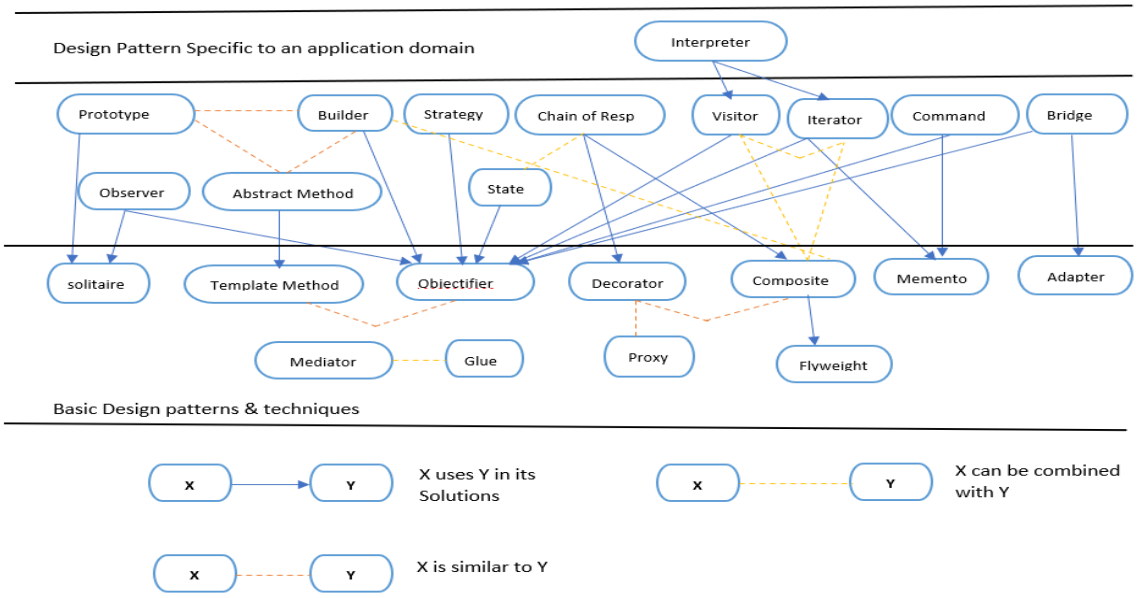


Figure 2.1 Zimmer's [30] Proposed a Solution Based Technique

2.3 Software Design pattern Recommendation

A well-defined classification scheme can aid in finding the group of relevant patterns and make accessible the process adapted to search the patterns for the given design problem. However, the selection or recommendation of the right design pattern from the list of relevant patterns is still an issue that has been addressed by many researchers in different domains.

It can be determined that selection of right design patterns is an issue, primarily, when an inexperienced developer in the domain where more than ten design patterns exist. Later, these issues have been addressed in consecutive studies [18]–[20], [32], [33]. Besides the support for the pattern selection, creation of a rank list of patterns and recommendation of appropriate patterns is also the essential features to resolve the design pattern selection process. Researchers have made their efforts to introduce several tools, recommendation systems, and prototypes to automate the design pattern selection process using different approaches. For applied approaches, we can summarize the existing efforts of researchers into three categories.

2.3.1 UML-based Automated Systems

D-K Kim and C.E. Khawand[34] describes an approach to select the design pattern by specifying its problem domain. They reported that the problem domain of design patterns could help to explain the known problems in the context. The authors described UML based

pattern specification language by using Role-Based Modeling Language (RBML) as the Meta-model to formalize the software design patterns. The structure of a Meta-Model is formulated through the class and collaboration diagrams to illustrate the structural and behavioral aspects of software and detect a design pattern. The structural similarity and scalability in terms of several design patterns and its variation are the main constraints in the implementation of UML-based approaches. Due to the structural similarity between design patterns, a Meta-model cannot specify the problem domain of design patterns. For example, in the case of the Gamma et al. design pattern catalog, the Meta-model for Strategy pattern looks closer to the Meta-model of the State design pattern. However, their goals are different. Subsequently, there are some design patterns such as Prototype, whose specification through Meta-model cannot be presented [34].

In their study, Hsueh et al. [35] also follow the UML-based approach and describe a case study with a limited number of design patterns. They consider four perspectives, such as activity-view, requirement-view, problem-view, and tradeoff-view, to find a systematic way to select a design pattern. They use UML notations to describe the design problems by incorporating the non-functional requirements besides functional requirements. The design pattern perspectives include,

- The Activity-View, to define how a design pattern can help a design activity,
- Requirement-View, to explore how a design pattern can improve the non-functional requirements,
- Problem-View, to investigate how a software design pattern prevent exceptions and help to resolve the design problems, and
- Tradeoff-View, to see how a software design pattern determine the design conflict.

The number of limited design patterns in a case study and selection of design patterns through a framework rather than automation are the main constraints [35].

In the domain of object-oriented development, researchers have introduced several tools to detect the pattern instance in the given source rather than to select a pattern for the given design problem. These tools can aid developers in finding the usage intensity of design patterns and their association with the internal (e.g., Coupling, cohesion, and complexity) and external quality attributes such as stability, understandability, flexibility, reusability, and effectiveness. Moreover, through these tools, a developer can investigate the declined quality of software. The common thing among all design pattern detection tools is the usage of UML

(i.e., Class diagram notations) to present the structure of patterns. Moreover, the techniques which are used to specify the patterns are different.

2.3.2 Ontology-based Automated Systems

Hasso and Carlson consider the problem definition part of design patterns and perform semantic analysis of sentences and on the base of words meaning to suggest the classification of design patterns. In their study, they perform semantic analysis using linguistic theory and consider the meaning of verbs for classification of design patterns. They claim that their proposed approach helps to select the right design pattern, which becomes in doubt due to lack of its evaluation and automation [36]. In two consecutive studies, Khoury et al. And Blomqvist address the automation of the selection of the right design pattern via their proposed ontology-based automated systems. For instance, in their study, Khoury et al., urge an ontological interface showed in Ontology Web Language (OWL) to support software designers to discover an eligible set of security design patterns. The proposed ontological interface implements a mapping between security requirements on one side and security bugs, security errors, and threats model on the other side of context [37]. Subsequently, Blomqvist [38] proposed a semi-automatic ontology construction approach named OntoCase, which relies on the use of ontology pattern to rank and select the right design pattern. In this work, the author defines criteria that include relation matching, class matching, density, centrality, and semantic similarity to rank the design patterns. In these studies, evaluation and automation factors are targeted. Subsequently, the computation time and specification in the case of the intricate design pattern collections are still the main issues in the evaluation of the proposed approach [39].

Researchers have introduced numerous recommendation systems to suggest the right pattern(s). For example, Gomes [40] applied a Case-Based Reasoning (CBR) approach to automate the selection of design patterns. The proposed CBR-based framework aims to select the patterns which can resolve a design problem and generate the new design. The proposed approach is implemented in the CASE tool REBUILDER, which can provide new CBR-based functionality. The cases of the proposed CBR-based framework are described in the form of a UML class diagram, which can be retrieved and updated with the support of REBUILDER. The retrieval of cases depends on the similarity between the structure of a pattern and the current design, along with the semantic distance of class and role names in the pattern.

In a recent study, Elizondo et al. [33] proposed an approach based on knowledge representation and information extraction to analyze the architectural pattern description to the specific quality attributes. The proposed method is automated using a computable model, which can be referred to as a prototype tool that helps inexperienced software architects. The proposed approach claims the applicability to focus on the quality attribute-oriented pattern selection and lack of limitation on the use of a predefined and closed pattern repository.

Researchers have introduced numerous recommendation systems to suggest the right pattern(s). Nuttapon Sanyawong et al. [41], by simplifying the difficulty of design pattern selection they proposed a problem-driven pattern recommendation framework for GoF design patterns. The framework uses a pattern usage hierarchy to guide a software designer through the process of matching domain-specific problems with their corresponding design patterns. The pattern usage hierarchy is constructed by collecting the usage cases of all patterns and categorizing them into a multilevel hierarchical structure based on the characteristics of the design problems solved in these usage cases. Based on the obtained hierarchy, questions are given to a designer to retrieve his/her intention, which is then matched with the pattern used for pattern selection. To verify the selection process and to clarify his/her understanding of the problem context, the designer is asked to instantiate the schema of the selected pattern, i.e., its participants and structural relationships among them, into the domain-specific class diagram of the problem under consideration. The proposed approach is not automated for selecting candidate pattern usage descriptions from the pattern usage hierarchy to reduce a user's effort in a pattern matching process.

Generally, ontology-based automated systems either require semi or fully formal specification of design problems and the patterns. Unlike the UML-based automated system, which mostly targets the solution domain of patterns and works in the defined context, the research community of ontology-based automated systems shows their interest in both the problem domain and the solution domain of design pattern. For example, Hasso and Carlson [36] consider the problem definition part of design patterns to perform the semantic analysis of sentences rather than the solution of patterns, while Gomes [40] target the solution domain of patterns. Like a UML-based automated system, ontology-based automated systems also work within the defined context and are not applicable across the catalogs.

2.3.3 Text classification based Automated Systems

In the domain design pattern selection, Hasheminejad and Galileo [32] follow the text classification approach and describe a two-step automated method to select design patterns

to the degree of similarity between the description of software design problems and design problem definitions of the fetched design patterns. The two steps of the proposed approach are Learning Software Design Patterns and Design Pattern Selection. In the first step, a machine learning model is trained for each software design pattern class. In contrast, in the second step, a candidate design class similar to a design problem is determined based on text categorization. Subsequently, the right design pattern is selected from the design pattern class suggested to the developers. This work has a few shortcomings. First, it needs a classifier for each design pattern class, which seems to be inadequate for the design pattern catalog, which has been either classified into many categories or obtained as interdisciplinary design patterns [10]. Second, the power of classifiers is highly related to sample size and sparse density, which are not considered in this work. Third, the research community only uses the Document Frequency (DF) technique for feature selection and evaluates the increase in performance of the proposed method. The Document Frequency (DF) is a simple technique, and it cannot overcome the multiclass problem in the feature selection process [42]. In a recent study, Wu et al. proposed a new ensemble method named ForesTexter to overcome the imbalanced text classification problem and benchmark the results on several widely-used imbalanced datasets. [42]. In a recent study, Wu et al. [43] proposed a new ensemble method named ForesTexter to overcome the imbalanced text classification problem and benchmark the results on several widely-used imbalanced datasets. Subsequently, the authors comparatively investigate the proposed ForesTexter method with standard random forest and different variants of SVM algorithms**.**

The feature selection methods are recommended as a pre-processing activity; however, the multi-class problem remains an issue to affect the precise learning of a classifier. In a recent study, Uysal et al. [42] and Hussain et al. [15], [18], [19] targeted the discriminative power of the feature selection method and addressed the multi-class problem effect on the performance of classifiers. In their study, Uysal et al. present an improved Global Feature Selection Scheme (IGFSS) and make it functional via four feature selection methods. Subsequently, the authors reported the effectiveness of the proposed scheme in the context of classification decisions via supervised learners. Hussain et al., leverage the IGFSS scheme and introduce a new feature selection method named Ensemble-IG to target the multi-class problem in the context of unsupervised learning.Unlike UML-based and ontology-based automated systems used for the identification of design patterns, the applicability of text classification based automated system is not limited to the predefined context and domain.

2.4 Text Classification and Applications

Text mining is the equivalent term of textual analysis or text data mining, which aims to retrieve meaningful and interesting information from unstructured texts. Text mining is an interdisciplinary domain of data mining, information retrieval, statistics, computational linguistics, and machine learning. The advancement in correlated fields (such as the generation of topic models through statistical analysis in the information retrieval domain), and availability of unstructured software engineering (SE) data [44] motivate the SE research community to use text mining to solve specific information retrieval problems. The work-flow of the text mining approach is divided into a) Information Retrieval (IR), b) Natural Language Processing (NLP), c) Data Mining (DM), and Information Extraction (IE). The essential tasks which are performed using text mining approaches are text classification and clustering, concept extraction, production of taxonomies, document summarization, and sentiment analysis. The use of text classification to introduce an automated system has been applied in certain domains, such as spam e-mail filtering [14], web page classification [15], sentiment analysis [16], author identification [17], and the design pattern selection [15], [18]–[20].

In the domain of machine learning and text mining, automatic text classification is performed either through supervised or unsupervised learning techniques. The former methods use class labels assigned to documents, and latter methods use attributes of the data and dis(similarity) measures to automate their learning. However, in both cases, some text preprocessing activities which are required to make precise learning[16]. The recommendation of a set of pre-processing activities is related to the interest of researchers and the context of the automated system. The following are some common activities related to the effectiveness of a software design pattern recommendation system

- Tokenization
- Construction of feature space with unique words
- Reducing feature space using feature extraction methods
- Supervised machine learning

The common aim of all these preprocessing activities is to reduce the dimensionality of the feature space to improve the performance of classifiers.

2.4.1 Tokenization

The tokenization activity is considered as some pre-processing activity used to form the basic unit of text, or it can be referred to as a segmentation process used to divide the text into words or sentences. In natural language processing, a single word (low-level tokenization) or group of words (high-level tokenization) is considered as a token. For example, the "Agile Software Developer" and ("Agile," "Software," "Developer") are examples of high-level and low-level tokenization. Each tokenizer (e.g., OpenNLP) performed some basic steps such as segmentation of text into words and handling of abbreviations and hyphenated words. Subsequently, some extra morphological and lexical information is required to handle the typographical structure of the words such as

- Isolating (e.g., Words of Mandarin Chinese can be divided into smaller units),
- Agglutinative (e.g., The words of Japanese and Tamil are divided into small units), and
- Inflectional (the boundaries of the morphemes of Latin are ambiguous and are not evident in grammatical meaning).

The effect of a tokenizer yields a feature space that consists of the number of words/tokens.

2.4.2 Construction of Feature Space with Unique Words

The feature space which is generated after the tokenization process has many meaningless words such as "are," "this," "and," "or," etc.., make noise in the textual data. The removal of stop words activity is performed to decrease such noise and reduce the feature space for the machine learning methods to produce the correct results. The development of a list of stop words based on the context of text languages becomes difficult. Consequently, the precision of the tool varied across the application. In the domain of removal of stop words methods, the existing efforts have been classified as

- The classic method [45],
- Zipf's Law [46],
- Term Based Random Sampling (TBRS) [47], and
- Mutual Information (MI) [48].

The word stemming process is performed to convert morphological forms of words (semantically related words) into their root words. There are several stemming algorithms designed on two basic assumptions that are a mapping of morphological forms (i.e., Same semantics) of words into their root words, and words without the same semantic are kept

separate. The existing efforts of a researcher to propose the word stemming algorithms can be categorized into

- Truncating Stemmers (Porters [49], Paice/Husk [50], Lovins, Dawson [51]),
- Statistical (HMM (Hidden Markov Model) [52], YASS (Yet Another Suffix Striper) [53], N-gram [54], [55]) Stemmers, and
- Hybrid Stemmers (Corpus-based [51] , context-sensitive based [54]).

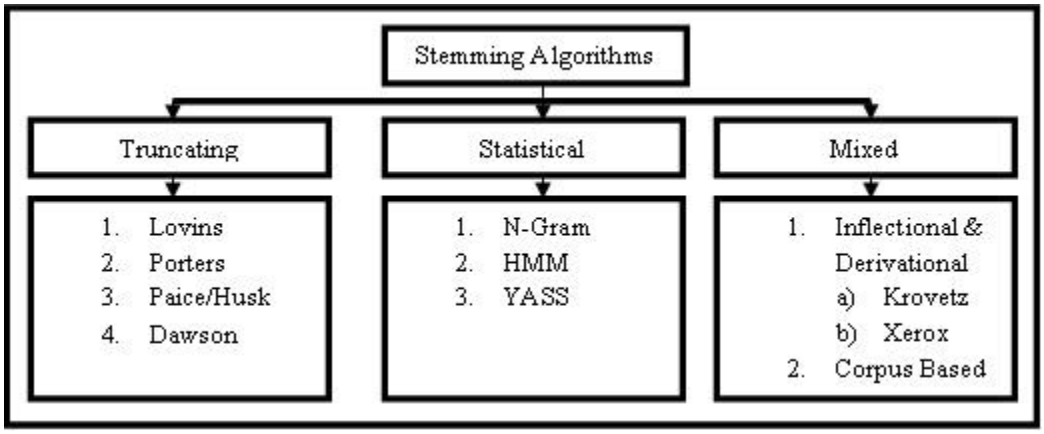


Figure 2.2 Stemming Algorithms Categories

In the pre-processing activities of certain automated systems, the truncating algorithms (especially Porter's algorithm) have been adopted by the research community due to their simplicity and lack of computational overheads. The truncating algorithms which are related to the removal of suffixes and prefixes (a.k.a affixes) use their lookup tables. The size of the lookup table, complexity, and less error rate are the factors to evaluate and compare the effectiveness of truncating algorithms. Porter's algorithm is one of the standard truncating stemming algorithms introduced in 1980; however, later on, it is modified. The lookup table of initial Porter's algorithm includes more than 1200 suffixes of as compared to Lovin's stemmer and Paice/Husk Stemmer, which provides for 294 and 120 suffixes, respectively. The number of suffixes target by Porter's and Dawson Stemmers is approximately the same. However, Porter's stemmer produces fewer error rates and complicated as compared to Dawson's Stemmer. Consequently, Porter's algorithm is used in the preprocessing phase of the automated system.

Lemmatization is a process based on morphological analysis of the tokens, i.e., grouping the numerous inflected forms of a word to categorize as a single item. It can also be defined as lemmatization methods attempt to map verb forms to infinite tense and nouns to a single form [56]. Lemmatization uses a vocabulary of words for morphological analysis and tries to remove inflectional endings, in consequence of that returning words to their dictionary

form. It checks to make sure things are correctly done by analyzing if query words are used as verbs or nouns. Lemmatization helps match synonyms by using a thesaurus so that when you search for "hot," the word "warm" will also match. Similarly, a search for "car" will return "cars" as well as "car."

2.4.3 Reducing Feature Space

In the vector space model, the undesirable effect of useless features must be reduced. As a result, feature extraction methods must be applied. The feature selection methods are grouped into wrapper embedded and Filter categories on the base of their interaction with classifiers to construct a feature set, which might be caused to increase the computation time. The wrappers and embedded methods are specific to the learning model as compared to filter-based methods [42]. Ogura et al. categorized the filter-based feature selection methods and referred to as one-sided and two-sided on the base of their characteristics. The one-sided feature selection methods assign a score greater than zero (i.e., Score ≥ 0) to a feature for membership class while assigning a score less than zero (i.e., Score <0) for non-membership classes. For example, the Odd Ratio is referred to as the one-sided feature selection method used to measure the membership and non-membership normalized values via its numerator and denominator [57]. The two-sided feature selection methods assign a score to feature, based on the combination of membership and non-membership in any class. For example, Document Frequency [57], Information Gain [58], [59] and improved Gini Index [58]. Subsequently, Tasci and Gungor classified filter-based methods into two categories and referred to as local and global, depending on unique or multi class-based score assigned to a feature [60]. In the text classification approach, the multiclass problem should be handled to improve the performance of classifiers. Gunal [61] applies the genetic algorithm and combine several filter methods and investigates the classification performance.

Feature extractions identify group features in the corpus data. They are creating a new, smaller set of features that still capture most of the useful information.

Word2Vec: - is based on the idea of deep learning that a fixed-length real-valued vector represents words. The word2vec tool transforms document operations into vector calculations of word vector spaces and uses features at the semantic level and improves the classification accuracy.

The semantic similarity of documents can be characterized by the similarity of the vector spaces in the word. Word2vec got the word vector can be applied to many natural language processing tasks, such as clustering, sentiment classification, find synonyms, and so on.

Word2vec tools can be implemented in two ways, namely CBOW (Continuous Bag-of-Words Model) model and the skip-gram model (Continuous Skip-gram Model).

Continuous Bag of Words (CBOW): predicts a current word based on its context. This latter corresponds to the neighboring terms in the window. In the process of CBOW, three layers are used. The input layer corresponds to the context. The hidden layer corresponds to the projection of each word from the input layer into the weight matrix, which is projected into the third layer, which is the output layer. The final step of this model is the comparison between its output and the word itself to correct its representation based on the backpropagation of the error gradient. Thus, the purpose of CBOW neural network is to maximize the following equation 3:

$$\frac{1}{\mathrm{V}}\sum_{t=1}^{V}\log \mathrm{P}\left(m_t|\, m_{t-\frac{c}{2}} \dots m_t + m_{t+\frac{c}{2}}\right)$$

Equation 2.1 Computing CBOW

Where V for vocabulary size, c is to the window size of each word.

Skip-Gram: The input layer corresponds to the target word, and the output layer corresponds to the context. Thus, Skip-Gram seeks the prediction of the context given the word instead of the prediction of a word given its context like CBOW. The final step of Skip-Gram is the comparison between its output and each word of the context to correct its representation based on the backpropagation of the error gradient. It seeks the maximization of the following equation 4:

$$\frac{1}{\mathrm{V}}\sum_{t=1}^{V}\sum_{\mathrm{j=t-c,j\neq t}}^{t+c}\log \mathrm{P}\left(m_j|\, m_t\right)$$

Equation 2.2 Computing Skip-Gram

Where V for vocabulary size, c is to the window size of each word. According to Mikolov et al. 1, each one of these models has its advantage. As an example, Skip-Gram is more efficient with small training data. Moreover, uncommon words are well presented. Nevertheless, CBOW is faster and have well performance with frequent words. However, learning the output vectors of CBOW and Skip-Gram represents one of the significant limits of these two models. Learning the output vectors can be a hard and expensive task. In this context, two algorithms can be used to address this problem.

The first algorithm is a Negative Sampling algorithm. This algorithm aims to limit the number of output vectors that need to be updating. Hence, a sample of these vectors is updated only based on a noise distribution. The second algorithm is the Hierarchical

SoftMax. This algorithm is based on the Huffman tree, a binary tree that presents all terms based on their frequencies. Then each step from the root to the target is normalized. According to Mikolov et al., each algorithm is better than the other according to the training data. For instance, negative sampling is more effective with low dimensional vectors, and it works better with frequent words.
Nevertheless, hierarchical SoftMax is better with uncommon words. In conclusion, using Word2Vec can be a hard and complicated task considering the different models (CBOW and Skip-Gram) and the used algorithms for training data (negative sampling and hierarchical SoftMax). Thus, in this paper, we will investigate the performance of these different models and algorithms to determine which ones are more efficient in the domain of topic segmentation

TF-IDF: TF-IDF (Term Frequency-Inverse Document Frequency) is a measure of the importance of a word in a document within a dataset and increases in proportion to the number of times that a word appears in the document. The TF-IDF is composed of two terms: The first computes the normalized Term Frequency (TF), which is. The frequency of a word appears in a document, divided by the total number of words in that document in equation 2.3. The second term is the IDF in Equation 2.3, calculated as the logarithm of the number of documents in the corpus separated by the number of documents in which the specific word occurs [62].

$$\mathrm{tfidf}(t, d, D) = \mathrm{tf}(t, d).\,\mathrm{idf}(t, D)$$

Where,

$$\mathrm{tf}(t, d) = \log(1 + \mathrm{freq}(t, d))$$

$$\mathrm{idf}(t, D) = \log\left(\frac{N}{\mathrm{count}(d \in D: t \in d)}\right)$$

Equation 2.3 Computing TF-IDF

TF-IDF is also one of the feature selection modeling methods used in software design pattern recommendations using text classification [15], [18]–[20], [63].

TF-IDF Weighted Word Embeddings: These feature extraction methods are embeddings that combined with the TF-IDF weight scores in each word of the sentence by multiplying the word embedding techniques with each word of TF-IDF score for the sentence of a given textual corpus. The technique is used for all the words in the sentence, and the result, of

multiplying the word TF-IDF by the word embedding, is accumulated and then divided by the accumulated TF-IDF scores of the words in a sentence. The obtained vector is used as the word embedding in the sentence. Paper [64] achieved results with several test cases. They took a case study with combination Word2Vec and TF-IDF without stop words and TF-IDF without stop words performed better than TF-IDF. They developed an algorithm that was able to acquire the top words and took the TF-IDF training and test models in determining the top words by counting the document frequency of each word and selecting the least frequent words followed by placed weights on these words giving them higher importance after the weighted sum of vectors applied. The fitted model using Word2Vec weighted by TF-IDF with the training data target and predicting the class labels, their results outscored the previous Word2Vec model in every instance.

There are two aims of the preprocessing phase of a text classification based automated system

- To transform unstructured text into a structured form via a vector space model, which can enable the supervised or unsupervised learning process, and
- To reduce the feature vector space dimensionality by removing noise in documents and enhance the precise learning of supervised and unsupervised learners.

Consequently, besides the recommendation of the pre-processing activities, the recommendation of a supervised and unsupervised learner is also important, because, in a latest empirical study with 179 classifiers, Delgado et al. [65] concluded that no one classifiers are better and it depends on the context of a real-world problem and working procedure selected classifiers. In the domain of text classification based automated systems, Naïve Bayes, Nearest Neighbor Classifier, Decision Trees, and Support Vector Machines (SVMs) are widely used, supervised learners. Subsequently, K-means, k-medoids, Fuzzy c-means, and agglomerative are widely used unsupervised learners, whose effectiveness is related to the selected distance measures such as Euclidean and Manhattan [65].

In a text classification based automated system, global filter-based feature selection methods are recommended. However, class-wise discrimination is the main issue that might be a cause of the multiclass problem, which is recently addressed by Uysal et al. and Hussain et al. in their studies. The recommendation of term weighting and feature selection methods for supervised and unsupervised learners based on the context of the conducted empirical studies. Consequently, the selected weighting method and feature method for a learner vary across the domain and context.

2.4.4 Supervised Machine Learning

In the machine learning domain, the text classification is exploited through supervised learning techniques. Recently, Delgado et al.[65] have evaluated the performance of 179 classifiers of 17 different families on 121 datasets retrieved from the UCL database to solve the real-world classification problem. In the domain of text classification based automated systems, there are several supervised learners. However, it cannot be decided which one is better since no classifier can get better results for all problems [65]. Consequently, it is recommended to find the best classifier for a new problem. In the domain of software design pattern automation using text classification, researchers have used and reported the classification performance of Naïve Bayes, K-NN, and Support Vector Machine (SVM) supervised learning techniques [15], [66]. We included Random Forest understudy for the promising reported result using a short text dataset enriched with data semantically similar to its words [67]. A brief description of these classifiers is given below.

2.4.4.1 Naive Bayes

The Naïve Bayes (NB) is a probabilistic classifier based on Bayes theorem. Naïve Bayes works on the assumption that all features are independent of each other and calculate a probability score by multiplying the conditional probabilities with each other. It is supposed that the pattern class L(pi) of pattern pi has some relation to words that appears in the description of Problem Definition of design patterns and design problems [68]. The Bayesian formula described in equation 2.4 yields the probability of a pattern class, given the words used in the description of patterns and design problems.

$$P(L_i|t_1,\ldots, t_{mi}) = \frac{p(t_1,\ldots,t_{mi} \mid L_k)\ p(\ L_k)}{\sum_{l \epsilon L} p(t_1,\ldots,t_{mi} \mid l)P(l)}$$

Equation 2.4 Naïve Bayes Formula

The prior probability depicts the probability that the design problem belongs to a class label before its known words. The conditional probability of words (includes in the description of patterns and design problems) given a class is described in equation 2.5

$$p\ (t_1 \ldots, t_{mi} \mid L_k\ ) = \prod_{j=1}^{mi}\ P\ (t_j \mid L_k\ )$$

Equation 2.5 Conditional Probability of Words in the Documents

The estimation of the probabilities of words P ($t_j|L_k$)is required to make the learning of the Naive Bayes classifier, which depends on their frequencies in documents (i.e., Design patterns) of the training set.

2.4.4.2 Random Forest

Random forests are a collection or ensemble of Classification, and Regression Trees (CART) is each tree of RF that depends on the values of a random vector sampled independently with the same distribution for all trees in the forest [69].

2.4.4.3 Support Vector Machines (SVM)

A Support Vector Machine (SVM) is considered as one of the most useful text classification algorithms, which looks for a decision surface and determines a margin through its closest data points. Usually, an SVM algorithm can separate documents of the training set into two classes, y=+1 for the positive and y=-1 for the negative class. Subsequently, the equation 2.6(1) is used to define a hyperplane at y=0 for the set of input vectors. Each input vector for the document d is represented as a count of words, as depicted in equation 2.6(2).

$$y = f(\underset{t_d}{\rightarrow}) = b_0 + \sum_{j=1}^{m} b_j t_{dj} \quad \text{----------(1)}$$

$$\underset{t_d}{\rightarrow} = t_{d1}, \ldots, t_{dM} \quad \text{----------(2)}$$

Equation 2.6 SVM Classifier Using Hyperplane

The SVM algorithm classifies a new document with $\underset{t_d}{\rightarrow}$ into a positive class if $f(\underset{t_d}{\rightarrow}) > 0$ otherwise into negative class.

2.4.4.4 K-Nearest Neighbors (K-NN)

The K-NN classifier is a type of instance-based learning. Classification is calculated from the nearest k training data in the feature space. Other methods such as the Bayesian classifier, K-NN does not count on prior probabilities. The main computation is the sorting of training documents to find the k nearest neighbors for the test document [70].

2.5 Related Works

There are some related works to efforts made for the classification of design patterns into an appropriate number of design pattern classes using text classification and to automate the selection process of the appropriate design pattern(s). For example, S. Hussain et al. [15], [18], [19] and paper[20], [71], consider the design pattern selection as an information

retrieval problem and used text classification approach, and the automated method to select the design pattern for the degree of similarity between the description of design problems and problem definitions of the retrieved design patterns.

S. Hussain et al.[18] fuzzy c-means an unsupervised learning technique as a core function to determine the resemblance of different objects and selection of the most appropriate pattern for a design problem. They used two samples that are Gang-of-Four (GoF) design pattern and spoiled pattern collection and three object-oriented related design problems. They used a fuzzy silhouette test, kappa (k) test, cosine similarity via ARGMAX Function to measure the effectiveness of their methodology. From the comparison results, they observed 11%, 4%, and 18% improvement in the performance of the proposed method as compared to supervised learning techniques of SVM, Naïve Bayes, and C4.5, respectively. Their paper didn't consider more design problems and design patterns to verify and generalize the applicability of their methodology.

Likewise, paper [63] they have used supervised learning technique 'Learning to Rank' to construct a ranking model for the selection of more appropriate design pattern(s) for a given design problem. Among numerous list-wise ranking algorithms (Coordinate Ascent, AdaRank, and LambdaMART), they used the LambdaMART ranking algorithm due to the existence of design patterns either as a variant or a spoiled for of canonical solution and ranking them to text relevancy with the given design problem. Their paperwork has some threat to the fact that they have used fewer design problem examples and design pattern collection during their evaluation.

Paper [15] tries bridging the disparity between the semantic relationship between design patterns (i.e., Documents) and features used in design pattern organization. In the construction of the Feature set phase, the DBN algorithm is applied to generate the features for the classification of documents into appropriate classes. Weighting method with the highest F-measure value is chosen as the best for the corresponding learning technique. They found a significant increase in the performance of classifiers with a feature set constructed through the proposed Approach (DBN) as compared to feature sets created through other feature selection methods. Their paper still does not show the applicability of the proposed approach to handle the multiclass problem by tuning the DBN parameters.

On the other hand, a proposed an approach where using text classification approach and unsupervised learners to achieve three objectives (1) to organize the patterns into the opposite numeral of pattern classes (i.e., Categories), (2) to select the appropriate design

pattern class for a design problem, and (3) to suggest and choose the appropriate design pattern(s) for a given design problem [19].

The study has some limitations. The first limitation is related to external validity where specific characteristics such as discriminative power of feature selection methods and sparsity in constructed VSM for a design pattern collection may affect the results. The second limitation is related to the internal validity of factors that could influence the results. They incorporate unsupervised learners with default parameters. However, the calibration of parameters of unsupervised learners could improve the effectiveness of the proposed framework. The third limitation is related to construction validity in terms of the creation of the VSM for the precise learning of unsupervised learners employed in the proposed approach. Though several experiments were performed to select the finest weighting and feature selection method, however, handling of a multiclass problem can improve the effectiveness of the framework. In this work, feature selection techniques are biased towards their discriminative power. Subsequently, the multiclass problem is not solved to include an equal number of features for each class into a constructed feature set.

Paper [72] uses the text classification approach as an information retrieval problem by generating a VSM of unigrams and topics to the catalog of patterns. The latent Dirichlet allocation topic model is used to analyze the textual descriptions of the patterns to extract the key topics and the hidden semantic of the corpus text. However, their proposed approach is not used for design patterns organization and lacks sufficient datasets. Similarly, paper [20] uses a text classification scheme, which begins with preprocessing, indexing of secure patterns, and ends up by querying SRS features for retrieving secure design patterns using the document retrieval model. Their work did focus only on secure design patterns catalogs, and the feature vector size is small.

Sanyawong et al. [66] proposed a text classification approach to help the user choose an appropriate design pattern category. Popular classification methods, d. H. Native Bayes, J48, k-NN, and SVM are used to classify given textual design problems into their design pattern group. The framework was rated with 26 case studies. They train the classifiers with the feature set constructed with IDF Transformation disabled, and TF Transformation enabled and vice versa. The classification results obtained from the J48 decision tree classifier are best performed from other classifiers.

2.5.1 Summary of Related Works

To sum up, a summary of related works and the ground needs to build upon are illustrated in Table 2.3 below.

Table 2.3 Related work summary

Ref No	Open issues	Problem solved	Not yet covered	What this thesis address
[18]	The existing automated design pattern selection methodologies are limited to a) Formal specification of design patterns. b) Lack of appropriate sample size	Better classification performance than supervised learners employed in [32]	a) It is limited to a small number of design problems(three). b) It is limited to one feature selection method, which is DF.	a) Uses more real design problem scenarios. b) Considers more than one feature selection method.
[19]	The previous existing automatic techniques are limited to a) The semi-formal specification b) The multiclass problem for the unsupervised classifiers. c) Small sample size to make precise learning.	The proposed method uses the Ensemble-IG feature selection method to solve a multiclass problem for a fuzzy c-means classifier.	It is limited to only a single word in the feature vector space model.	Considers semantics of connecting words instead of a single word in the feature.
[15]	a) Semantic relationship between design patterns and the features which are used for the organization of design patterns.	Proposed an approach to construct a more illustrative feature set via deep learning named Deep Belief Network (DBN) and aid in improving the classification performance of the classifier.	a) Default values adjusted for the parameter of DBN b) Limited to Micro-F1 performance measure.	It considers macro-F1 measures to evaluate the effectiveness of the proposed method in terms of class discrimination.

Table 2.3 Related work summary [continued from the previous table]

Ref No	Open issues	Problem solved	Not yet covered	What this thesis address
[72]	It is challenging to select a fit design pattern for a given design problem.	Proposed a solution to use a topic model to avoid representing each pattern and each problem as a vector of unigrams only and considers the semantic similarity between the problem scenario and patterns.	The method based on unigrams features or topics.	It Considers one vector(word2vec) and weighting score per word (TF-IDF) for feature set building and demonstrates their impact on the models.
[20]	a) Having less security knowledge during software design b) Difficult to choose a pattern from extensive security design pattern collection	The proposed approach provides an easier way to select a suitable security pattern using software requirement specification.	The proposed method depends on the similarity of the two vector space models.	Adopt supervised learning models to train design pattern descriptions and design problems.
[71]	Having a difficult way to have a supporting tool that automatically suggests a design pattern for a given problem to the developer.	Proposed an approach based on text retrieval on building a VSM for the GoF design patterns	Have low-quality design problem scenarios to evaluate the method.	Uses real design problem scenarios from authentic books.
[63]	a) The need for formal specification b) Need Precise learning through training the various classifiers.	Proposed a method use learning to rank supervised learning technique and similarity with the description of the given design problems.	Limited to term weighting schemes for feature selection text pre-processing.	Prioritize well-performed feature extraction methods that are exploited via supervised learners.
[66]	Difficult to determine the first-level category and to consider which design pattern should be applied to the problem	Text classification is used to assist category selection processes providing the best appropriate group to a given question.	The low-quality feature set when creating vector space model for text classifications and few testing data sets	Relatively extensive test data set to evaluate text classification algorithms.

CHAPTER THREE

3 Research Methodology

In this chapter, the methodology for the research is described examining the current system to get an understanding of proposed method development, through comparing and contrasting available development tools to select them for the proposed system, defining the design of the proposed system and testing method. The following sections in this chapter explain and justify the methodology used in conducting the study on software design pattern recommendation.

3.1 Systematic Literature Review

The study conducted a systematic literature review to identify the characteristics of software design patterns and to refine the state-of-arts as well as the knowledge scope in the text classification automation for software design pattern trends.

The systematic literature review was prepared according to the procedure used by [72] because they present a systematic literature review in an attempt to understand the application of different text classification techniques reviewed accordingly in the paper. The authors used the search process, which was created in different searching steps to collect the relevant studies.

3.1.1 Definition of research questions for software design pattern recommendation

To consume an overview of a research area and, at the same time, identify the quantity of research and result available within it, we asked the following questions and thereby set a goal from which research papers were identified for software design pattern recommendation.

- Which journals databases do include papers on software design patterns?
- What are the most investigated techniques for the problem?
- What types of text classification methods are addressed for software design pattern recommendation?

3.1.2 Conduct literature search

In this study, we use a search string consisting of four parts, namely, Software Design Pattern Recommendation, Text Classifications, and Machine Learning concatenated using the operator "AND." The leading search string is done by connecting the following keywords through logical "OR." The query strings and search scope are as follows:

- **Query String**
  - (Automatic **OR** Recommendation) **AND** (Software Design Pattern **OR** Design pattern **OR**) **AND** (Text Classification **OR** Text Mining **OR** Machine Learning)
- **Timespan**: 2011-2019
- **Language**: English
- **Reference Type**: Journal, Conference, Workshop, Symposium, Books

To search-relevant papers on software design recommendation and text classifications, we use four electronic databases for relevant studies all of which indexed software design pattern papers.

1. IEEE Xplore
2. ScienceDirect
3. SpringerLink
4. ACM Digital

These databases are the most appropriate to the field of the Text Classification study [72]. They also provide suitable and accessible search engine mechanisms which are simple for the spontaneous search of resources in their database based on the combination of search term/string. With the specified reference databases, we search for a paper between 2011 and 2019. We choose 2011 as a starting date because previous work [33] has surveyed software design pattern automation using text classification has brought a comprehensive overview of existing work. We set the end date in 2019 because the search started in 2019.

3.1.3 Refining the Search Result

Inclusion Criteria

1. The study should be in the area of software design recommendation using machine learning.

2. The method proposed by every study should be using Text Classification or Text Mining.
3. The automation of the process focuses on the efficiency of software design pattern classification and selection.

Exclusion criteria

1. A study is not written in the English language.
2. A publication year is not between 2011 to 2019.

As illustrated in Fig 3.1 we adopted the search process to identify and select studies from a different number of venues. A total of 3,403 results were retrieved from the automated searches based on targeted query strings. Based on the inclusion and exclusion criteria, only 230 papers were found after eliminating 3,173 irrelevant papers and thereby maintaining studies that were related to this work. We then remove 169 identical papers and 61 unique papers identified. Then, we filtered by reading abstraction and the introduction section of each paper. Also, if it becomes confusing to judge from the abstract and introduction, we read the conclusion. If it is still difficult to deduce the relevance of the paper, we keep for the next step. In this step, we obtained 42 research papers for further study.

Finally, the full content of each paper was examined and made a final decision about each paper. This step resulted in 23 research papers.

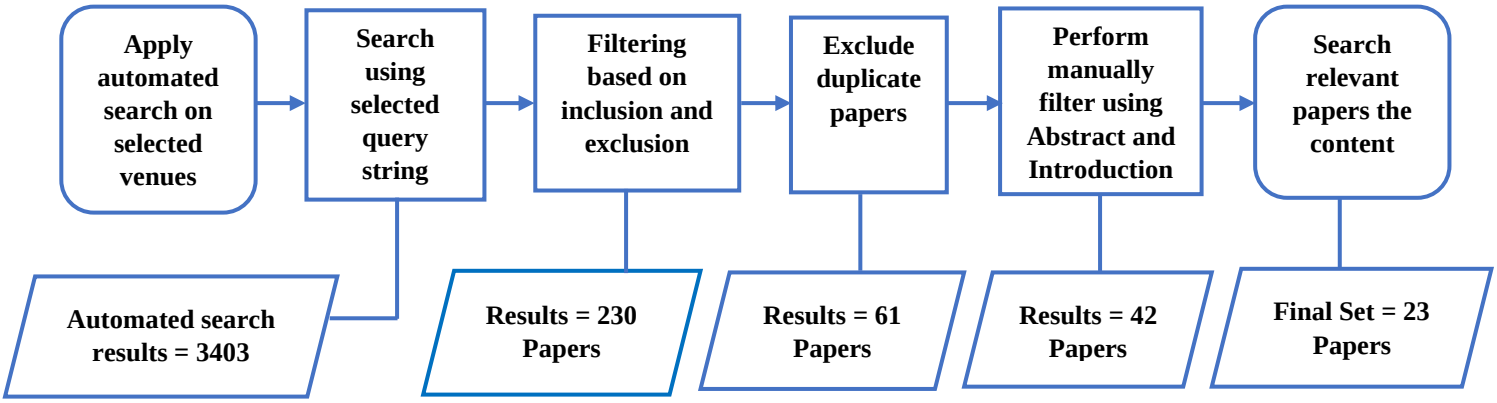


Figure 3.1 Identification and Selection of Primary Studies

3.2 Data collection

In this study, software design pattern recommendations using text classification approach identified from the literature review. In the following steps, the identification, formulation,

and analyses of sample text datasets for software design pattern recommendations are explained.

3.2.1 Building a Dataset

The objectives of this study are to recommend software design pattern(s) for specific design problem, So, it needs to build a dataset. This dataset needed because there is no published or annotated dataset for this purpose. For data preparation, we used tools such as MS Excel and others. The process of building the sample dataset for software design pattern recommendations consists of three main steps,

1. Gathering GoF software design patterns and real design problem descriptions of their textual data.

2. Preparing, filtering, or consolidating gathered data into two datasets (i.e. Training and Test Dataset), and

3. Annotating the dataset.

3.2.1.1 Training Dataset

In software engineering, some groups of experts from different domains have categorized the design patterns on the base of their intent, motivation, and applicability. The domain of patterns, similar community interest, the number of design patterns, and pattern classes are the main factors behind the rationale to select a design pattern collection. We consider one often used design pattern collection in our proposed evaluation model.

The GoF Design Patterns are described to target the particular issue that occurs in the context of object-oriented software development. The GoF collection includes 23 object-oriented design patterns, which are divided into three groups, named Creational, Structural, and Behavioral. This study uses the problem domain of GoF patterns as described on [10] for training the dataset to train the classifiers. Only textual sentences from the problem domain section are given to the classifiers, with figures being omitted.

3.2.1.2 Test Dataset

To evaluate the proposed method, we retrieved 38 real design problems from different sources [9], [73], [74]. We retrieved the design problems which have been addressed and mapped according to expert opinions, which can aid us in investigating the authenticity of

the proposed solution. The reference of data resources, authors who address and map the problems, and the involvement of the community is shown in Table 3.1. We have described design problems in Appendix B to evaluate the effectiveness of the proposed method.

Table 3.1 Resource information about design problems for GoF Collection

Reference of Resource	Authors	Involved Participants
Bouhour et al. [9]	Bouhour, Leblance, and Percebois	126
Shalloway et al. [73]	Shalloway and Trott	3
Shvets [74]	Shvets	1

3.3 Text Classification

Text classification or text categorization means discovering their category or topic from a set of predefined classes, e.g., 'Creational,' 'Structural' or 'Behavioral.' Text classification is an essential field within natural language processing. The other meanings of text classification/categorization are explained in, somewhat different activities are referred to as automated text classification. In this thesis, we mean the automatic assignment of category labels to documents, where the categories available are known in advance. Others say the discovery of the groups, or the discovery process and the assignment of category labels more often known as clustering. An even broader definition of the word includes any method of grouping documents by categories, that is, both the variations mentioned above.

Text classifiers can be one of supervised or unsupervised machine learning algorithms. We have decided to focus on the classic supervised machine learning algorithms Naïve Bayes, RF, k-NN, and Support Vector Machine, which has seen increasing popularity for text categorization. The aim of the supervised learner (i.e., Classifier) is to classify the text documents into their predefined classes based on their learning from the contents of all documents [15]. In case of supervised learning, suppose, we have a training dataset $D=(d_1, d_2, d_3, \ldots, d_n)$ of N documents that are classified into n labels and described as $L = \{l_1, l_2, \ldots, l_n\}$. The supervised learning model is represented by the equation 3.1.

$$f: D \rightarrow L \qquad f(d) = l$$

Equation 3.1 Supervised Learning Model

3.3.1 Type of Text Classification

Text classification can be listed into many of them in various ways, i.e., multiple labels, multi-class, single label, binary label, etc. Multiclass is a classification task with more than two classes. Each sample can only be labeled as one class. We approach software design classification for a particular design problem as a multi-class problem with three categories. That is where one design problem would be “Creational,” “Structural,” and the other “Behavior.”

3.4 Document Representation

Our document is represented in the vector space model, the most popular model used in Information Retrieval and Text Classification[21]. It is an underlying document representation that used in many applications, including document ranking in search engines. In-text classification, all documents in the experiment are represented by the same model, the vector space model.

3.4.1 Preprocessing

The goal of the preprocessing is to reduce the feature set size and data sparsity. Each design pattern description and design problem scenario is processed through the following activities

- **Tokenization and Normalization:** The token is a sequence of characters and does not include delimiters such as punctuation marks and spaces. Tokenization is the process where the text is splitting at the delimiters into tokens (words). Then all the words are normalized by transferring them into lowercase.
- **Stop-Word Removal**: It is the process of elimination of non-descriptive words like linking verbs and pronouns. Stop words are considered noise; they increase the size of the Vector Space Model and do not contribute to the retrieval process.
- **Lemmatization**: this process is used to convert a word to its base form. This is done by considering the context of the word to its meaningful base form.

3.4.2 Feature Extraction Methods

Feature extraction methods based on state-of-the-art text mining; techniques are performed to reduce the dimensionality of the feature space by removing unimportant and meaningless terms. The methods involved in selecting a subset of relevant features that would help in identifying creational, structural, and behavioral from the dataset and can be used in the

modeling of the software design recommendation using text classification techniques. The TF-IDF and Word2Vec feature extraction used in this study because they are better and popular feature extraction methods used design pattern recommendation and text classification problems. Along with TF-IDF and Word2Vec features extraction methods ,we used a combination of these two feature extraction methods in this study methodology.

Word2Vec Weighted By TF-IDF: the documents are represented as the TF-IDF weighted sum of the Word2Vec vectors corresponding to each word. This method calculates weights to each word based on its frequency within the document in Word2Vec to get weighted sums of word vectors.

3.5 Evaluation

The recommendation of software design patterns to their categories can be treated as a classification problem where the number of classes is the number of design patterns included in the corpus of design pattern descriptions.

3.5.1 Construction of Training and Test Set

To evaluate the proposed method, we consider widely-known design patterns collection by considering three parameters

- Application domain,
- Number of design patterns, and
- Number of design pattern classes.

The brief description of our training and test data set in this study is as follows.

3.5.1.1 Gang-of-Four Design Patterns

The aim of these design patterns is to describe simple and elegant solutions to specify recurring design problems in the context of object-oriented development. These design patterns aid those developers who are struggling for the flexibility and reuse in their object-oriented software. The authors of the GoF design patterns introduce 23 design patterns by considering different aspects related to objects and classes, such as abstraction, coupling, and cohesion. Subsequently, these design patterns are grouped into Creational, Structural, and Behavioral categories (a.k.a Design pattern classes). The performance of models is evaluated on separate and independent test dataset of documents which are not used during the training/learning process. The test set consists of GoF design problem descriptions.

3.5.2 Evaluation Metrics

The evaluation of supervised models consists of a set of measures to assess the effectiveness of the text classification algorithms, which are used for the recommendation of software design pattern categories. The metrics that are used for this study are listed below.

3.5.2.1 Classification Accuracy

Classification Accuracy can be calculated using the ratio number of accurate predictions to the total number of input samples. It only works when an equal number of examples belonging to each class [74]

$$\text{Accuracy } = \frac{\#\{\text{ Correct Predictions}\}}{\#\text{Total predictions made}}$$

Equation 3.2 Accuracy Calculation

3.5.2.2 Precision (Positive Predictive Value)

Precision measures the predicted value actual, and it shows how many times the model predicts true. Precision answers what proportion of identifications was correct [74]. To Precision calculated using the equation:

$$\text{P} = \frac{\sum_{c=1}^{\text{N}} \text{TP}_\text{C}}{\sum_{c=1}^{\text{N}} (\text{TP}_\text{C} + \text{FP}_\text{C})}, \qquad Micro - Averaging$$

Equation 3.3 Calculate Precision of the Classifier

Where n is a number of design patterns, $\sum_{c=1}^{N} TP_C$ is the number of correctly recommended design patterns. While $\sum_{c=1}^{N} FP_C$ is the number of incorrectly recommended patterns. Macro-average precision could also be used, but we selected the micro average level due to the imbalance in the testing data used in the evaluation is that the number of design problems is not equal across the design patterns (training set).

3.5.2.3 Recall

Recall answers what proportion of actual positives was correctly identified [74]. Recall calculated using the equation:

$$\text{P} = \frac{\sum_{c=1}^{\text{N}} \text{TP}_\text{C}}{\sum_{c=1}^{\text{N}} (\text{TP}_\text{C} + \text{FN}_\text{C})}, \qquad Micro - Averaging$$

Equation 3.4 Calculate Recall of the Classifier

3.5.2.4 F Measure

The F measure (F1-score or F-score) is a measure of a test's accuracy, and that can be defined as a weighted harmonic mean of the precision and recall of the test. F measure is more useful than accuracy when the dataset contains imbalanced class distribution [75]. This score is calculated using this equation:

$$F = \frac{2 * (\text{Precision} * \text{Recall})}{\text{Precision} + \text{Recall}}$$

Equation 3.5 F Measure for the Classifier

In the equations 3.3-3.5, N is the number of design pattern classes decided to evaluate the performance of learning techniques. For example, in the case of GoF design pattern collection, N is 3. The term TP is the number of design problems that are correctly identified for each design pattern class. FP is the number of design problems that are incorrectly identified for each design pattern class, and FN is the number of design problems that are missing from each corresponding design pattern class. The values of P, R, and F are computed using equations 3.3-3.5, respectively.

3.5.2.5 Confusion Matrix

A confusion matrix is a technique for summarizing the performance of a classification algorithm. The number of correct and incorrect predictions are summarized with count values and broken down by each class. It gives insight not only into the errors being made by the classifier but, more importantly, the types of errors that are made. So, this study uses a non-normalized confusion matrix for the three design pattern classes in the case of GoF collection. The following Table 3.1. shows a sample format of a confusion matrix with 3-classes.

Table 3.2 Sample Format of a Confusion Matrix for Design Pattern Classes

		Predicted values		
		Behavioral	**Creational**	**Structural**
Actual Values	**Behavioral**	**True Behavioral**	False Creational	False Structural
	Creational	False Behavioral	**True Creational**	False Structural
	Structural	False Behavioral	False Creational	**True Structural**

CHAPTER FOUR

4 Proposed Solution for Software Design Pattern Recommendation

4.1 Overview of the Proposed Method

The employment of design patterns in the software development process is considered as an alternative approach for software refactoring to address the declined quality of the software system. In recent studies, the authors have reported the developer's interest and user's perception to use the design patterns and their implications concerning certain aspects. The existence of more than one classification scheme to organize the patterns of the same domain inconsistencies and overlapping of patterns on the online repositories and lack of developer's knowledge motivates us to introduce an approach that can aid in organizing the design patterns. The authors of the widely used Gang-of-Four (GoF) design patterns quoted the design pattern selection(recommendation) process as a problem when a developer is working with a collection of 23 design patterns, especially when a developer is unfamiliar (or lack of knowledge) with the target pattern catalog. Hasheminejad and Jalili, Hussain et al., and Birukou recommend the scan of the intent of design patterns to understand the classification scheme to narrow the search and selection of patterns, which are more relevant to a design problem.

Due to an increase in the development of the software design pattern in certain domains published either through books or online repositories, we consider design pattern recommendation as an information retrieval problem and look at the capacity of text classification [15], [18], [19], [63], [66], [72] to exploit the proposed method.

The proposed method aims to use supervised learning techniques to model the design patterns of the target catalog, which are recorded using either the same or different classification schemes. Subsequently, the proposed method uses a systematic way to select the right design pattern(s) for a given design problem on the base of text relevancy. There are three aim of the proposed method

1. To model the design patterns of a target pattern catalog via supervised learning technique using a description of their problem definition (i.e., Intent of design patterns)

2. To recommend a correct pattern class (i.e., group of similar patterns) for a given design problem based on the constructed supervised model.
3. To recommend the right design pattern(s) for a given real design problem using similarity technique. We applied the text classification steps on the description of the problem domain part of the design patterns (shown in Appendix A) and description of the design problem (shown in Appendix B). The proposed approach is formulated in four steps, which are described in Figure 3.1.

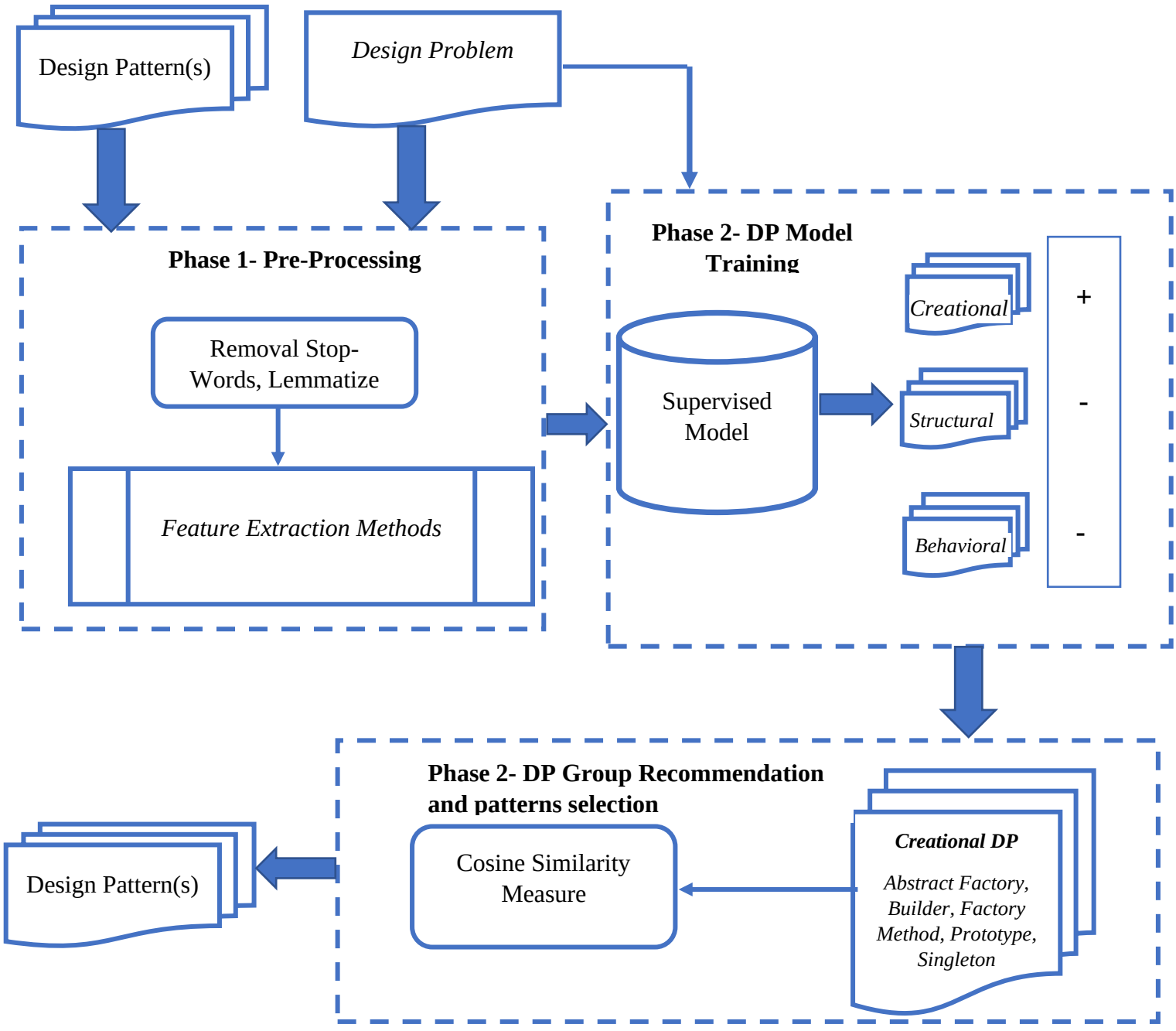


Figure 4.1 Overview of the proposed framework

4.2 Proposed Text Preprocessing

The first phase of the proposed approach is preprocessing. In this phase, a set of activities is performed to automate the text classification of the target text.

The precision of supervised learning is highly related to the selection of pre-processing activities used to map the unstructured text into a structured form. The list of preprocessing activities is shown in Figure 4.2. The description of design problems and problem definition part (i.e., Intent) of design pattern is the target text (i.e., Inputs) of the proposed method

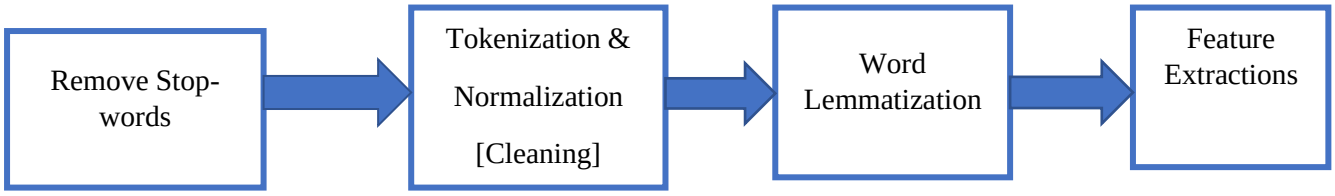


Figure 4.2 Set of Preprocessing Activities

The outcome of the first phase is in the form of features as vectors.

4.2.1 Tokenization

Tokenization is the process of breaking a document into a sequence of characters into pieces (a single word, a pair of words or sentences) called tokens. Some characters are removed at this stage, such as symbols and punctuation marks.

4.2.2 Normalization

List of tokens that were generated from the tokenization process then subjected to normalization or cleaning of data. Normalization is usually done on documents to remove some unnecessary words. The most common is stop-words removal. Stop words are the high-frequency words that appear in the text without having any important information (e.g., prepositions, conjunctions, etc.). On the other hand, high-frequency words in the text are said to have little information to distinguish different documents, and also words occurring at lower frequency are possible of non-significant relevance and should be removed from the documents.

4.2.3 Word Lemmatization

After normalization, the main activity in the proposed method is word lemmatization. The reason we use lemmatization over stemming is to take into consideration the context of the design problem words to determine which is the intended meaning the user is looking for.

4.3 Proposed Feature Extraction Methods

4.3.1 TF-IDF

The proposed method encompasses this feature extraction method activity to find the important features words of terms ‘t’ in a document.’ This study has used the Term Frequency-Inverse Document, as it is the most common feature extraction method. TF-IDF has been used for a similar problem. Hussain et al. [19], [20], [66] find it the most efficient weighting method. The aim of the Term Frequency Inverse Document Frequency (TF-IDF) weighting method is to assign the proportion of frequency of words in the document and inverse proportion to the number of documents in which word occurs at least one time as its weight.

4.3.2 Word2Vec

In this study, the proposed Word2Vec method performs word to vector modeling on design problems and definitions part of the design patterns corpus. Word2Vec brings similar semantic features that help in the software design recommendation process using text classification. This study used skip-gram by default because skip-gram has the highest semantic accuracy but at the cost of time efficiency [64]. Our work settled on a default vector size parameter for the majority of our experiments for time efficiency.

4.3.3 Word2Vec weighted by TF-IDF

The proposed feature extraction in the study has not only to perform feature extraction based on single methods. However, also, we proposed a combination of some of these methods together like Word2Vec weighted by TF-IDF. This proposed feature extraction method is the weighted average, where the weights are TF-IDF and allow us to capture that some words in a sentence are naturally more valuable than others.

Table 4.1 Pseudo code to extracting features using Word2Vec weighted by TF-IDF

Pseudocode-4.3.3 Feature Extraction using Word2Vec with TF-IDF
Input:
PC = {Set of problem domain description of design patterns in the given Pattern Collection and Design Problem Scenarios}
Procedure Begin:
1. Apply a feature extraction method such as TF-IDF to constructed VSM. 2. Apply word embedding method, such as using Word2Vec.

3. Create new lists to store for TFIDF-W2V vectors, word vectors new array of given shape and type, filled with zeros, numbers of words with a valid TF-IDF vector.
4. Multiple vectors from step 2 with Step 1 and add to the list of empty created word vectors with shape and type, filled with zeros.
5. Sum step 4 to step 1 on the new empty created list of the number of words TF-IDF vector.
6. Averaging by dividing step 4 with step 5.
7. Merging vectors from step 6 into step 3.
8. For each sentence in the data set, repeat steps 3,4,5,6, and 7 until all possible assessments have done.
9. For each word in a sentence, repeat steps 4,5,6 and 7 until all possible assessments have done.

Procedure End

Output:

Word2Vec Vectors Weighted By TF-IDF

4.4 Proposed Model Construction Using Supervised Learning

The proposed approach is based on the text classification approach. Subsequently, it is exploited by supervised learning techniques for model construction. The model used to organize the design pattern groups with correct expert opinions, and later on, it is used to narrow down the search for the required pattern within their group.

Consequently, the organization of design patterns via supervised learners can aid in overcoming the problems related to pattern searching. Our proposed method in this study considers four supervised learning techniques that show remarkable results on papers[15], [70]. In a text classification supervised machine learning approach, it is recommended to test several models regardless of their theoretical performance because their accuracy is dependent on the feature set and training dataset. The details of the supervised learning techniques, which are used in this study are discussed in chapter two.

4.4.1 Pseudocodes for the Organization of Design Patterns via Supervised Learners

The pseudocode-4.4.1 aims to describe the flow of activities that are performed to achieve the target objective, which is the implementation of the proposed method for the organization of design patterns.

Table 4.2 Pseudocode for Organization of Design Patterns via Supervised Learners

Pseudocode-4.4.1 Pattern Organization (PC, USC, k, N, M)

Input:

PC = {Set of problem domain description of design patterns in the given Pattern Collection}

SL = {Set of Supervised Learners under this study, such as SVM, K-NN, NB, RF}

k = The number of groups/classes to organize the pattern catalog, which is recommended through expert opinion, For example, in the case of GoF (k=3)

N = The number of design patterns in a collection, for instance, in the case of GoF collection, the value of N is 23.

M = The number of unique words

Procedure Begin:

1. Perform the first three activities of the preprocessing phase of the proposed framework to construct a VSM with N design patterns and M Unique words.
2. Apply a feature extraction method such as TFIDF to constructed VSM.
3. Apply word embedding method, such as using Word2Vec.
4. Apply word2vec weighted with TF-IDF
5. Apply supervised learning techniques to organize N design patterns by constructing a model.
6. Repeat Steps 2, 3, and 4 separately until all possible assessments have done.

Procedure End

Output:

The outperform supervised learner with the best feature extraction method for the organization of the design patterns catalog (i.e., PC).

4.5 Identification of Design Pattern Class

In this study, a design pattern class refers to a group of patterns recommended via the supervised learners of the proposed method for a given design problem. The number of design pattern classes (according to expert's opinion) depends on the decision of supervised

learning techniques used in the proposed method. Subsequently, each block inside the rectangle (labeled as a Phase 2) refers to a design pattern class, while + and – symbols denote the decision of the corresponding learning techniques for the given design problem. The + symbol describes the closeness of a design problem with patterns (i.e., Members) of corresponding candidate design pattern class (i.e., groups).

4.5.1 Pseudocodes for Identification of Appropriate Design Pattern Class

The aim of this pseudocode is to describe the flow of activities that are performed to achieve the target objective that is the implementation of the proposed method for the determination of appropriate pattern classes for the sample of given design problems. The identification of proper pattern class for a given design problem via the proposed method is described through pseudocode-4.5.1 as follows

Table 4.3 Pseudocodes for the Identification of Appropriate Pattern Class

Pseudocode-4.5.1 Determing_Pattern_Class_For_Problem (PC, DP, USC, k, N, M)

Input:

PC = {Set of problem domain description of design patterns in the given design pattern Collection}

DP = {Set of descriptions of the given design problems described in the context of target pattern collection}

SL = {Set of Supervised Learners understudies, such as SVM, K-NN, NB, RF)}

k = The number of groups/classes to organize the pattern catalog, which is recommended through expert opinion, For example, in the case of GoF (k=3)

N = The number of design patterns in a collection, for instance, in the case of GoF collection, the value of N is 23.

M = The number of unique words

Procedure Begin

1. Perform the first three activities of the preprocessing phase of the proposed framework to construct a Support Vector Model (SVM) with N+1 (i.e., N design patterns and one target design problem) feature vectors and M Unique words.
2. Perform the Step-2 to Step-3 of pseudocode-4.4.1 to design problems.
3. Apply into the trained model to determine the DP class.

4. Apply evaluation criteria F-measure to assess the performance of each supervised learner with k (e.g., In the case of GoF pattern collection, the value of k is 3)
5. Apply the evaluation criteria measures to assess the performance of the supervised learners with corresponding their feature extraction methods
6. For each supervised learner, select the best feature extraction method.
7. Select the outperform supervised learner.

Procedure End

Output:

The outperform supervised learner with the best weighting method to identify the right pattern class (i.e., Category) for the given sample of design problems (which are already mapped into appropriate categories according to expert opinions). For example, in the case of Design problem 1(DP-1), the Behavioral pattern class is the right choice according to expert opinion.

4.6 Design Patterns Recommendation for a Given Design Problem

The supervised learning techniques used to narrow down the pattern search via the group of patterns on the base of their similarity and dissimilarity employed through distance measure Cosine. Through their category, the pattern search becomes narrow. Subsequently, the selection of the right pattern(s) from the recommended class (i.e., Pattern class for the given design problem) is still required. This phase of our proposed method aims to suggest the design pattern(s) using similarity measures. A similarity measure is suggested to compute the degree of closeness between given design problems and patterns of suggested pattern class. In the proposed method, we recommend the Cosine Similarity measure to compute the degree of closeness between design problems and patterns of suggested pattern class. The main factors behind the recommendation of Cosine Similarity measures are.

i. The Cosine similarity measure is very efficient in assessing the sparse vectors (i.e., Vectors, which have mostly zero values) as compared to other measures. In this regard, we consider two documents and describe them in the vector form by depicting the frequency of their independent terms.

ii. The heterogeneous description of published patterns [33] varies in length, which motivates us to suggest the use of Cosine Similarity (CS) measure because this measure is independent of document length [76].

iii. Since the proposed approach is introduced by considering the design pattern classification and selection as an information retrieval problem, the Cosine similarity measure is suggested for the text classification based applications [18], [19], [32], [76].

iv. The cosine similarity measure relied on the orientation of the feature vector rather than the magnitude and normalized the inner product of vectors.

The aim of the Cosine Similarity measure in the third phase of the proposed method is to find the degree of closeness between the given design problem and the N design patterns of suggested pattern class (in the second phase of the proposed method). Firstly, the term weighted vectors for $Pattern_i$ (iϵN) and Problem are described through equation 4.1

$$W(d) = (w(d,t_1), w(d,t_1), \dots, w(d,t_n))$$

Equation 4.1 TF-IDF Weighted Vectors

Secondly, the correlation between design problem and design patterns of the suggested pattern class is calculated using equation 4.2.

$$CS_i = \sum_{j=1}^{N} w(Pattern_i, t_j) \times w(Problem, t_j)$$

Equation 4.2 Cosine Similarity

Subsequently, after computing the degree of closeness between the given design problem and the N design patterns of suggested pattern class, we use the Argument of the Maxima (Arg Max) function to find the most significant output.

Consequently, we used the Arg Max function as the first similarity option, which aids in recommending the right design pattern (with the highest cosine value) from the list of N patterns. Equation 4.3 can help in finding the k^{th} design pattern (as of right design pattern) to a developer.

$$k = \arg\max CS_k$$

Equation 4.3 Arg Max Function to Select Software Design Pattern

4.6.1 Pseudocodes for the Recommendation of Design Patterns in their Groups

Pseudocode-4.6.1 aims to describe the flow of activities that are performed to achieve the target objective that is the selection of right design patterns for the given sample of the given design problems. The implementation procedure for the selection of the right design pattern(s) for a given design problem via the proposed approach is described as the pseudocode-4.6.1 as follows.

Table 4.4 Pseudocodes for Recommendation of Design Patterns

Pseudocode-4.6.1 Pattern_Selection_for_Problem (Pattern Vectors, Problem Vectors

Input:

Pattern_Vectors = {Set of the feature vectors for design patterns of a pattern class recommended for the target design problem}

Problem Vector = Describe the feature vector for the target design problem

Procedure Begin

1. The equation 3.4 can be used to measure the closeness of patterns with the given design problems.
2. The similarity technique is applied to select the right design pattern(s).
3. Equation 4.3 described the ARG maxima method, which aids in selecting the right design pattern for a given design problem on the base of the cosine similarity value.

Procedure End

Output:

The selection of right design pattern(s) from the patterns list (Pattern Vectors) of determined pattern class to content similarity for the given design problem (i.e., Problem Vector).

4.7 Model Evaluation and Testing

Text classification automation for software design pattern recommendation is evaluated based on how well they perform compared to the true design pattern group information of design problem text documents. Using documents with known design pattern group information in our test set, we can evaluate a classifier performance in various performance metrics. The percentage of the correctly classified document is the main point of the evaluation. However, there are still some important information measures besides the percentage of correctness.

4.7.1 Supervised Learning Techniques

For assurance and the verification of the supervised learner's ability on the software design patterns is a significant phase in the proposed method. The underlying concern of the machine learning model is to find an accurate estimation of the generalization error of the trained models on finite datasets. Because of the reason this study used different performance evaluation metrics appropriate for models, these metrics are confusion matrix, accuracy, precision, recall, and F-Measure, as discussed in chapter three.

4.7.2 Design Pattern Groups

In software engineering, some groups of experts from different domains have categorized the design patterns on the base of their intent, motivation, and applicability. The domain of patterns, similar community interest, the number of design patterns, and pattern classes are the main factors behind the rationale to select a design pattern collection. We consider one often used design pattern collection in our proposed evaluation model.

4.7.2.1 Gang-of-Four (GoF) Design Pattern Collection

The GoF Design Patterns are described to target the particular issue that occurs in the context of object-oriented software development. The GoF collection includes 23 object-oriented design patterns, which are divided into three groups, named Creational, Structural, and Behavioral. This study uses the problem domain of GoF patterns as described on [10] for training the dataset to train the classifiers. Only textual sentences from the problem domain section are given to the classifiers, with figures being omitted.

4.7.3 Real Design Problems

To evaluate the proposed method, we retrieved 38 real design problems from different sources [9], [22], [77]. We retrieved the design problems which have been addressed and mapped according to expert opinions, which can aid us in investigating the authenticity of the proposed solution. The reference of data resources, authors who address and map the problems, and the involvement of the community is shown in Table 4.5. We have described design problems in Appendix B to evaluate the effectiveness of the proposed method.

Table 4.5 Resource information about design problems for GoF Collection

Reference of Resource	**Authors**	**Involved Participants**
Bouhour et al. [9]	Bouhour, Leblance, and Percebois	126
Shalloway et al. [77]	Shalloway and Trott	3
Shvets [22]	Shvets	1

CHAPTER FIVE

5 Implementation and Experimentation

5.1 Overview

The purpose of this chapter is to address the implementation procedure of the proposed method for software design pattern recommendation by using the methodology discussed in chapter three. Also, implement the proposed solution in chapter four. This study follows an experimental approach to determine the performance of the classifier and recommendation of software design pattern for a given design problem using different feature extraction techniques. The first implementation of the experiment is feature extraction methods using TF-IDF, the second is using Word2Vec, and the third using Word2Vec weighted by TF-IDF for building a supervised classifier model.

5.2 Implementation Environment

This study used several development tools and packages to implement the proposed solution. This study uses python programing language for implementing and experimenting with each proposed solution from the data preprocessing to the model building phase. Also, to evaluate the implemented proposed classifiers model. Python used because of its programming language of choice for developers, researchers, and data scientists who need to work in machine learning models. Table 5.1 shows the list of tools and python packages with their version and description, which is used in this study for implementation.

Table 5.1 Description of the Tools and Python Packages

Tools		
Tools	Version	Description
Anaconda Navigator	1.9.12	Allows us to launch development applications and easily manage conda packages, environments, and channels without the need to use command-line commands.
Spyder	4.1.3	It is an interactive development environment for the Python

		language with advanced editing, interactive testing, debugging, and introspection features.
Python	3.7.7	Easy to learn, powerful programming language to develop a machine learning application.
Microsoft Excel	2016	Used data preparation tasks in cleaning filtering and sorting the gather data.
Python Packages		
Scikit-learn	0.20.1	A set of python modules for machine learning and data mining. This study uses it for feature extraction and training and testing model. The name of the package is called sklearn.
Pandas	1.0.3	High-performance, easy-to-use data structures, and data analysis tools. This study uses it for data reading, manipulation, writing, and handling the data frame.
NumPy	1.15.4	Array processing for number, strings, and objects. This the study uses it for handling converting the text to numeric data for features and training and testing the model.

Gensim	3.8.0	Python library for topic modeling document indexing and similarity retrieval with large corpora. This study uses it for the constricting word2vec model.
NLTK	3.4.5	This study uses it for tokenization.
Matplotlib	3.1.3	Publication quality figures in python. This study uses it for data and results visualization.

5.3 Deployment Environment

The tools which are discussed above in section 5.2 have been deployed on a personal computer equipped with a processor Intel® Core™ i7-6500U, CPU 2.50GHz, 2.59GHz Core(s), 8 Gigabyte of physical memory, 900 Gigabyte hard disk storage capacities. The operating system is Windows 10, 64-bit with the x64-based processor.

5.4 Dataset Descriptions and Insights

To organize the dataset, we use the description of the problem definition domain of GoF software design patterns. The performance of the proposed method is evaluated using two text corpus; one of them includes the textual descriptions of 23 design patterns from the catalog of GoF design patterns. Each pattern document includes the intent, applicability, participants, and collaborators. The GoF books were used to prepare a rich description document to each pattern, including the pattern of distinctive words. In contrast, the other corpus includes 38 real design problem scenarios collected from various design patterns books, as discussed in section 4.7.3 of chapter four.

The training dataset is built from the corpus of GoF problem domain texts that are manually annotated with the specified design pattern name and design pattern class or group that has been labeled with a class of Creational, Behavioral and Structural. Design pattern groups with their labeled class only used for modeling of design pattern group recommendations. The training dataset contains the following four columns.

- **dp _id:** is a design pattern id for each design patterns
- **dp_class:** is a design pattern category in which each design pattern belongs to
- **dp_problem_domain** a design problem description for each design pattern
- **dp_name**: the name of each design patterns

A portion of the labeled dataset is set aside for testing and validating the trained model to assess its generalization ability of the study. The test dataset is a corpus of sample real design problem texts that are annotated with their design pattern group(class) and design pattern name.
The test dataset contains the same number columns as the training expect with the column names

- **dp_problem_id:** is a design problem description for each design pattern
- **dp_class:** is a design pattern category in which each design problem belongs to
- **dp_problem:** a design problem description for each design pattern
- **dp_name**: the name of the specific design pattern

We create an implementation code to analyze and get an insight into the corpus text data used for the training and testing of supervised learners. The loaded dataset using panda is used throughout the whole implementation of this study as shown in Figure 5.1

```
data= pd.read_csv('C:/Users/Error/.spyder-py3/DP_Recommendation
                  '/Corpus_Dataset/Test_Data_Sample_Two.csv',
                  names=['DP_Problem', 'DP_Class'],encoding = 'utf-8',
                  header = None)
```

Figure 5.1 Loading Test Dataset

We calculate the number of characters in each training and test dataset on their problems text data. In Figure 5.2 characters have been counted on test design problems dataset by using python built-in-function str.len to get lengths of each design problem in the column. We perform the 'groupby' operation on the 'DP_Class' label and prints the average character length across the labels.

```
# Get char count
data['character_cnt'] = data['DP_Problem'].str.len()
num_char = data.groupby('DP_Class')['character_cnt'].mean()
print(num_char)
```

Figure 5.2 Counted Characters in the test dataset

To calculate the number of words in the design problems text data in the rows. We used python built-in-function str.len that returns the length of pandas' series values on the 'DP_Problem' column index label and we perform the 'groupby' operation on the 'DP_class' label and prints number of words in the test dataset as shown in Figure 5.3

```
#Get word count
data['word_counts'] = data['DP_Problem'].str.split().str.len()
num_words = data.groupby('DP_Class')['word_counts'].mean()
print(num_words)
```

Figure 5.3 Counted Words in the test dataset

Using the ratio of the number of characters and the number of words in the design problem test dataset. we perform the 'groupby' operation on the 'DP_class' label and prints the average character length per word across the labels as shown in Figure 5.4

```
#Average Character Length per Word
data['characters_per_word'] = data['character_cnt']/data['word_counts']
avg_char = data.groupby('DP_Class')['characters_per_word'].mean()
print(avg_char)
```

Figure 5.4 Average characters length per word in the test dataset

We have imported stop words from NLTK, which is a basic NLP library in python. To do this, we use "apply" and "lambda" functions of the panda library along with the split function of python. we perform the 'groupby' operation on the 'DP_class' label and prints number of stop words in the dataset of the problem domain of design pattern shown in Figure 5.5

```
#Stop Words in the Labels
stop = stopwords.words('english')
data['stopwords'] = data['DP_Problem'].apply(lambda x: len(
    [x for x in x.split() if x in stop]))
num_stop_words = data.groupby('DP_Class')['stopwords'].mean()
print(num_stop_words)
```

Figure 5.5 Counted Stop words in the dataset

The same implementation procedures are made for testing datasets.

5.5 Suggestion for Design Pattern Group

5.5.1 Preprocessing Implementation

The preprocessing phase is implemented; the study used python programming language and modules. To perform the full implementation of the methods, we perform the following joint

activities, which are importing important libraries packages and load the dataset file to the disk using the code in Figure 5.6. The preprocessing method uses a python NLTK library.

5.5.1.1 Implementation of Removing Irrelevant Character and Stop words

This implementation aims to remove the irrelevant character and stop words in the corpus data. We implement methods to convert characters into lowercase, remove punctuation, and stop words shown in Figure 5.1. For the removal of stop words, we used the NLTK libraries corpus. For the removal of irrelevant characters, we use str.replace built-in functions of the panda's library. The code for the removal of irrelevant characters and stop words of the dataset is shown in Appendix C.2.

```
# remove punctuation
def remove_punctuation(data):
    data = data.str.replace('[^\w\s]','')
    return data
# Converting to lowercase
def text_lowercase(data):
    data = data.apply(lambda x: " ".join(x.lower() for x in x.split()))
    return data

# remove stopwords function
def remove_stopwords(data):
    stop = stopwords.words('english')
    data = data.apply(lambda x: " ".join(x for x in x.split()
                                         if x not in stop))
    return data
```

Figure 5.6 Removing Irrelevant character and Stop words

5.5.1.2 Implementation of Tokenization

This implementation aims to tokenize corpus text data into tokens or words for later use of feature construction in feature extraction methods. As shown in Figure 5.6, this method performs using the RegexpTokenizer class that imported from nltk.tokenize python library. The code for the removal of irrelevant character and stop words of the dataset is shown in Appendix C.2.

```
def tokenization(data):
    #Instatiate Tokenizer
    tokenizer = RegexpTokenizer(r'\w+')
    data = data.apply(lambda x: tokenizer.tokenize(x))
    return data
```

Figure 5.7 Implementation of Tokenization

5.5.1.3 Implementation of Word Lemmatization

The aim of this implementation is used to lemmatize to get useful information from the corpus data. We used WordNetLemmatizer python implementations from the NLTK stem library. This method used is shown in Figure 5.9. The code for the word lemmatization of the dataset is shown in Appendix C.3.

```
# lemmatize string
def lemmatize_word(data):
    lemmatizer = WordNetLemmatizer()
    #word_tokens = word_tokenize(text)
    # provide context i.e. part-of-speech
    data = data.apply(lambda x: " ".join([lemmatizer.lemmatize(word) for word
                                          in x.split()]))
    return data
```

Figure 5.8 Implementation of Word Stemming and Lemmatization

5.5.2 Feature Extraction Implementation

The aim of this implementation is used to implement TF-IDF and Word2Vec feature extraction techniques in the proposed solutions. We used the python Scikit-learn module for TF-IDF and python Gensim module for Word2Vec.

5.5.2.1 Implementation of TF-IDF

Tf-IDF learned the vocabulary and document frequency of each word in the training and testing data. We used a *TfidfVectorizer* class of sci-kit learns package to implement TF-IDF. This class converts a textual dataset to a matrix of TF-IDF features vectors using the *fit_transform ()* method of the class shown in Figure 5.8. The study experiment with different hyperparameters and the best parameter is applied for the construction of TF-IDF vectors. Subsequently, TF-IDF vectors are used as input for modeling and to weight Word2Vec, which is one of the feature extraction methods.

```
############TF-IDF Implementation ##########

vectorizer = TfidfVectorizer(
             use_idf = True,
             smooth_idf = False,
             norm = None,
             decode_error = 'replace',
             max_features = 5000,
             min_df = 1,
             max_df = 1
             )
tf_idf_train_data = vectorizer.fit_transform(final_X)
```

Figure 5.9 Implementation of TF-IDF

5.5.2.2 Implementation of Word2Vec

We use Word2Vec feature modeling with genism to import and instantiate word2vec class with the necessary parameters. The cleanses vocabularies from problem domain design patterns and design problems after pre-processing are used to model word2vec to extracts semantic features that help in text classification.

The aim of this implementation is used for word embedding as shown in Figure 5.9

```
#word Embedding Word2Vec

w2v_data = final_X # Input final raw cleaned data into the pipeline
splitted = []
for row in w2v_data:
    splitted.append([word for word in row.split()])     #splitting words
# Set values for various parameters for the model
num_features = 100    # Word vector dimensionality
min_word_count = 1   # Minimum word count
num_workers = 4      # Number of threads to run in parallel
context = 5          # Context window size
downsampling = 1e-3  # Downsample setting for frequent words
train_w2v = Word2Vec(splitted, workers=num_workers, \
            size=num_features, min_count = min_word_count, \
            window = context, sample = downsampling)
words = list(train_w2v.wv.vocab) # Trained Words in the w2v model
train_w2v.save('train_save.bin') # Save the Model
```

Figure 5.10 Implementation of Word2Vec

A shown in Figure 5.10, we have set different hyperparameters that are suitable for model construction in the given problem domain of the proposed method. Subsequently, the trained words in the word2vec model are saved for later use.

5.5.2.3 Implementation of TF-IDF weighted Word2Vec

This implementation follows the pseudocode in chapter 4 of section 4.3.3 aims to add TF-IDF weights to each word based on its frequency within the vocabulary in the word2vec model after performing preprocessing steps. Figure 5.10 shows word2vec weighted by TF-IDF.

```
####W2V weighted By TF-IDF Implementation ####
tf_w_data_train = []
tf_idf_train_data_arr = tf_idf_train_data.toarray()
i = 0
for row in splitted:

    vec = np.zeros(10)
    temp_tfidf = []
    for val in tf_idf_train_data_arr[i]:
        if val != 0:
            temp_tfidf.append(val)
    tf_idf_sum = 0
    print(temp_tfidf)
    print(row)
    for word in row:
        count = 0
        try:

            count += 1
            tf_idf_sum = tf_idf_sum + temp_tfidf[count-1]
            vec = vec + (temp_tfidf[count-1] * train_w2v.wv[word])
       except:
            pass
    vec = (float) (1/tf_idf_sum) * vec
    tf_w_data_train.append(vec)
    i = i + 1
```

Figure 5.11 Implementation of TF-IDF weighted Word2Vec

5.5.3 Machine Learning Models Implementations

To build machine learning models using the proposed algorithm, we used a python sklearn supervised learning library package that is shown in the Figure 5.12 and followed the conventional method that is importing the essential library packages for training and testing supervised learners.

```
#import necessary modules
import DP_Feature_Extraction as dfe #Importing DP Preprocessing Library
import DP_test as dptest #Importing DP Testing Data Library
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sn
from sklearn.svm import LinearSVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn import metrics # Metrics for Learners Performance
```

Figure 5.12 Loading Important Library

Once the documents were cleaned and then featured extracted using TF-IDF, Word2Vec, TF-IDF weighted Word2Vec used as training and testing vectors to feed into the supervised classifiers. We named these feature vectors as 'X,' 'Y,' and 'Z' respectively with their three feature extraction methods and the label which contains the design patterns class in which the design problems belongs to set as 'y' as a label that is shown in Figure 5.13.

```
#training Data
X = dfe.tf_idf_train_data #  _data via TF-ID_dp_intent
Y = dfe.w2v_data        #  _data via Word2Vec_dp_intent
Z = dfe.tf_w_data_train   #  _data via tf&Wv_dp_intent
#target values
y = dfe.y_tr.values.tolist() #Dp_classes Labels for Training dataset
```

Figure 5.13 Loading Training and Testing Data

The supervised algorithms trained separately with each feature vectors such as 'X,' 'Y' and 'Z' are extracted with the three methods in the proposed solution. To train using the *fit()* method with the appropriate set parameters for each classifier instantiate the class.

To implement SVM, we used *LinearSVC ()* of the sklearn package to build the model. This classifier is instantiating with a basic parameter like *OVR* to classify the multi-class dataset and other default parameters. For imbalance data set SVC of the sklearn package, we can set the values for parameter C automatically by setting class_weight = 'balanced'. The balanced argument automatically weighs classes to increase the penalty for misclassifying minority classes to prevent them from being "overwhelmed" by the majority class.

```
#-------------------SVM via TF-IDF-----------------#
#Load the SVM Model
svc_model_tfidf = LinearSVC(multi_class = 'ovr', penalty = 'l2',
                            loss = 'squared_hinge', dual = True,
                            C= 0.01, class_weight = 'balanced',
                            verbose = 0, max_iter = 10)
#Train the algorithm
pred_SVC_tfidf= svc_model_tfidf.fit(X,y)
final_svc_pred_tfidf = pred_SVC_tfidf.predict(x_data_test)  #Predicted Output
```

Figure 5.14 Implementing SVM Classifier

To implement K-NN supervised learners, we instantiate KNeighborsClassfier class by assigning default parameters as shown in Figure 5.15.

```
#--------------------KNN Classfier via TF-IDF--------------------#
#create object of the classifier#
neigh_tfidf = KNeighborsClassifier(n_neighbors=2,leaf_size=30,p=1)
#Train the algorithm#
pred_knn_tfidf= neigh_tfidf.fit(X, y)
#Predicted Output#
pred_kneighbors_tfidf = pred_knn_tfidf.predict(y)
```

Figure 5.15 Implementing K-NN Classifier

To implement naïve Bayes classifiers, we create a naïve bayes object that is a method of the sklearn naïve_bayes. We train the algorithm on training data and predict using the testing data that is shown in Figure 5.16

```
#--------------------Naive Bayes Classfier via W2V-TFIDF---------------#
 #create an object of the type GaussianNB
gnb_tfidfw2v = GaussianNB(priors=None, var_smoothing=1e-03)
#Train the algorithm on training data and predict using the testing data
pred_GaussianNB_tfidfw2v = gnb_tfidfw2v.fit(X.toarray(),y)
#Predicted Output
pred_NB_tfidfw2v = pred_GaussianNB_tfidfw2v.predict(z_data_test)
```

Figure 5.16 Implementing Naive Bayes Classifier

The random forest classifier is implemented using the *RandomForestClassifier* () method of the sklearn package is used to train the classifier. This classifier fits several decision tree classifiers as declare in the n_*estimators* parameter that is shown in Figure 5.17

```
 #--------------------Rf_via_w2vec-------------------#
 #create an object of the type of RF
RFC_W2V = RandomForestClassifier(
min_samples_leaf=1,
min_samples_split=2,
min_weight_fraction_leaf=0.0,
n_estimators=10,
n_jobs=2,
random_state=0,
verbose=0)

pred_rfc_W2V = RFC_W2V.fit(Y,y) #Train the model using the training sets
#Predicted Output
pred_randomForestClassifier_W2V=pred_rfc_W2V.predict(y_data_test)
```

Figure 5.17 Implementing Random Forest Classifiers

5.6 Recommendation of Design Pattern from their Group

This phase is to find the right design pattern for the sample of given design problems using the cosine similarity measure. In this regard, the following implementation of Figure 5.18 is performed to find the closeness between design problems and design patterns of the recommended pattern group via the supervised classifiers.

```
def calculate_csm(csm_dp_problem, csm_dp_name) :
    csm_dp_problem, csm_dp_name = (csm_dp_problem, csm_dp_name)
    if len(csm_dp_problem) < len(csm_dp_name)
    else (csm_dp_name, csm_dp_problem)
    csm_dp_name = csm_dp_name[:len(csm_dp_problem)]
    try:
        assert any(csm_dp_problem)
        assert any(csm_dp_name)
    except AssertionError:
        print('Couldnt compute Cs of these vectors !')

    return np.round(np.inner(csm_dp_problem, csm_dp_name)/
                    (LA.norm(csm_dp_problem)*LA.norm(csm_dp_name)), 1)
```

Figure 5.18 Implementation of Cosine Similarity of Two Vectors

5.7 Model Testing and Evaluation

The trained classifiers are tested using design problems test data. All supervised classifiers in the study used to implement a predict () method of scikit-learn packages that returns the predicted labels of design pattern groups or class. We set test vectors as x_data_test, y_data_test, and z_data_test using via predict method these vectors created separately using feature extraction methods, as shown in Figure 5.19.

```
#Testing Data
x_data_test = dptest.X_test # Testing DP problem Data with TF_IDF
y_data_test = dptest.Y_test # Testing DP problem Data with W2V
z_data_test = dptest.Z_test # Testing DP problem Data with TF_IDF_W2V
```

Figure 5.19 Testing Data for the supervised Models

The study used a classification_report (), accuracy_score (), and confusion_matrix () methods, which are used to evaluate the performance of the built models using the predict method. These methods give a result of evaluation metrics such as accuracy, precision, recall, and F1-score value of the model under testing.

CHAPTER SIX

6 Result and Discussions

The proposed method is employed in the context of widely used design pattern collection in different domains and many design problems that have been mapped and resolved by the research/developer community. The details of design pattern collections and related design problems are discussed in chapter four. In this regard, we present a case study that includes a design pattern catalog and a set of related problems.

In this chapter, the effectiveness of the proposed method is reported in terms of determining the right design patterns for the sample of given design problems. Subsequently, the results are discussed within the context of the target design pattern collection and sample of related design problems. The analysis of the case study is structured into two sections based on the target objectives of the proposed method to recommend the design pattern group and to select the correct design pattern.

6.1 Case Study (Gang-of-Four Design Patterns in the Domain of Object-Oriented Development)

6.1.1 Determination of Design Pattern Class

The proposed method aims to find the right design pattern class for the sample of given design problems. The experiment is performed with a sample of 38 design problems to evaluate the effectiveness of the proposed method. Moreover, it is assumed that the sample should include all design patterns from each design pattern class (such as three pattern classes in GoF pattern collection).

6.1.1.1 Dataset Insight

Characteristics of the dataset determine to achieve the intended solution for a given classification problem. Training dataset has three design pattern class each consists of unique instances, that is 5 Creational, 7 Structural, and 11 Behavioral. The test dataset has 6 Creational, 15 Structural, and 17 Behavioral with duplicate instances. Before applied preprocessing activities of the proposed method, we perform basic data checks to gain information on our training and test datasets as described in section 5.4. As a result, we extracted important information from the raw text data of the test dataset, as discussed below.

Character Length

In Figure 6.1 the output shows that the design problem with the creational group has a lower character count on an average, as compared to behavioral and the creational design problems in the test dataset.

```
DP_Class
Behavioral    518.285714
Creational    327.571429
Structural    382.857143
Name: character_cnt, dtype: float64
```

Figure 6.1 Character Length in each Design pattern Class

Word Count

The output shows that the negative behavioral design pattern group has the highest average word count, suggesting that design problems that need to be addressed by behavioral design patterns tend to have a complexity of the problem in the software design process. This can be useful for separating the 'sentiment' labels.

```
DP_Class
Behavioral    87.928571
Creational    54.142857
Structural    63.642857
Name: word_counts, dtype: float64
```

Figure 6.2 Number of Words in each Design pattern Class

Average Character Length per Word Number of words

The output shows that design problems with creational design pattern groups have the highest average character length per word than others.

```
DP_Class
Behavioral    6.114893
Creational    6.225132
Structural    6.105939
Name: characters_per_word, dtype: float64
```

Figure 6.3 Average character length in each design pattern class

Number of Stop Words

The output shows that design problems with Behavioral design pattern group have the highest number of stop words than others.

```
NUMBER OF STOP WORDS

DP_Class
Behavioral    36.428571
Creational    21.285714
Structural    26.714286
Name: stopwords, dtype: float64
```

Figure 6.4 Number of Stop words in each Design pattern Class

The above information may implicate that, the length of the characters, number of words, and number of stop words in most software design problems in which their solution occurs in behavioral design pattern groups can be used for more complex needs in software systems. Sample design problems that grouped within the creational patterns their highest average character lengths per word numbers, lowest on word count, lengths of characters, and several stop words, they have the least clarity on describing the problem issues due to their focus on object creation mechanisms.

6.1.1.2 Feature Extraction Results

The features extraction process using the three methods TD-IDF, Word2Vec, TF-IDF weighted Word2Vec produced seven different sets of features vectors from the dataset. These features vectors are uses in the training of SVM, K-NN, NB, and RF models. Table 6.1 shows the extracted features vector size for the training dataset.

Table 6.1 Results of the Extracted Features Vectors Size

Features Extraction method	**Features vectors size**
TF-IDF	6877
Word2Vec	2300
TF-IDF +Word2Vec	2300

6.1.1.3 Models Evaluation Results

The result of the test dataset accuracy score for SVM, K-NN, NB, and RF models based on extracted feature vectors are presented in tables separately below.

Table 6.2 SVM Model Accuracy for Each Features Extraction

Feature Extraction Method	Accuracy on Sample-Dataset (%)	F1-Score Weighted Avg (%-Micro and Macro Avg)
TF-IDF	73.7	74.0
Word2Vec	76.3	78.8
TF-IDF + Word2vec	81.6	**81.6**

The results in Table 6.2 shows the accuracy scores for the SVM model based on features with corresponding test data test. The lower average accuracy of 74.0% results recorded on TF-IDF the feature model and the higher accuracy 81.6% resulted using the word2vec weighted by the TF-IDF feature method.

Table 6.3 K-NN Model Accuracy for Each Features Extraction

Feature Extraction Method	Accuracy on Sample-Dataset (%)	F1-Score Weighted Avg (%-Micro and Macro Avg)
TF-IDF	60.5	**62.2**
Word2Vec	53.0	53.0
TF-IDF + Word2vec	58.0	59.0

The results in Table 6.3 shows the accuracy scores for the K-NN model based on features with corresponding test data set. The lower average accuracy of 53.0 % results recorded on TF-IDF the feature model and the higher accuracy of 62.2% resulted using TF-IDF weighted by the TF-IDF feature method.

Table 6.4 NB Model Accuracy for Each Features Extraction

Feature Extraction Method	Accuracy on Sample-Dataset (%)	F1-Score Weighted Avg (%-Micro and Macro Avg)
TF-IDF	71.1	70.2
Word2Vec	73.7	72.2
TF-IDF + Word2vec	78.9	**79.1**

The results in Table 6.4 shows the accuracy scores for the NB model based on features with corresponding test dataset. The lower average accuracy of 70.2% results recorded on TF-

IDF the feature model and the higher accuracy 79.1% resulted using word2vec weighted by the TF-IDF feature method.

Table 6.5 Random Forest Model Accuracy Score for Each Features Extraction

Feature Extraction Method	Accuracy on Sample-Dataset (%)	F1-Score Weighted Avg (%-Micro and Macro Avg)
TF-IDF	50.0	**50.1**
Word2Vec	47.4	44.4
TF-IDF + Word2vec	47.0	45.0

The results in Table 6.5 shows the accuracy scores for the RF model based on features with corresponding test dataset. The lower average accuracy of 44.4% results recorded on word2vec the feature model and the higher accuracy 50.1% resulted using the TF-IDF feature method.

F1-score metric selected to compare the models based on the features because it's more useful than accuracy when the dataset has imbalanced classes as described in chapter three. We follow the procedure described in the pseudocodes in chapter four and use the performance measure criterion to find the best feature extraction methods for each supervised learner. According to the F1-score measure criterion, the general evaluation results in the selection of best feature extraction methods for each supervised learner is shown in Figure 6.5.

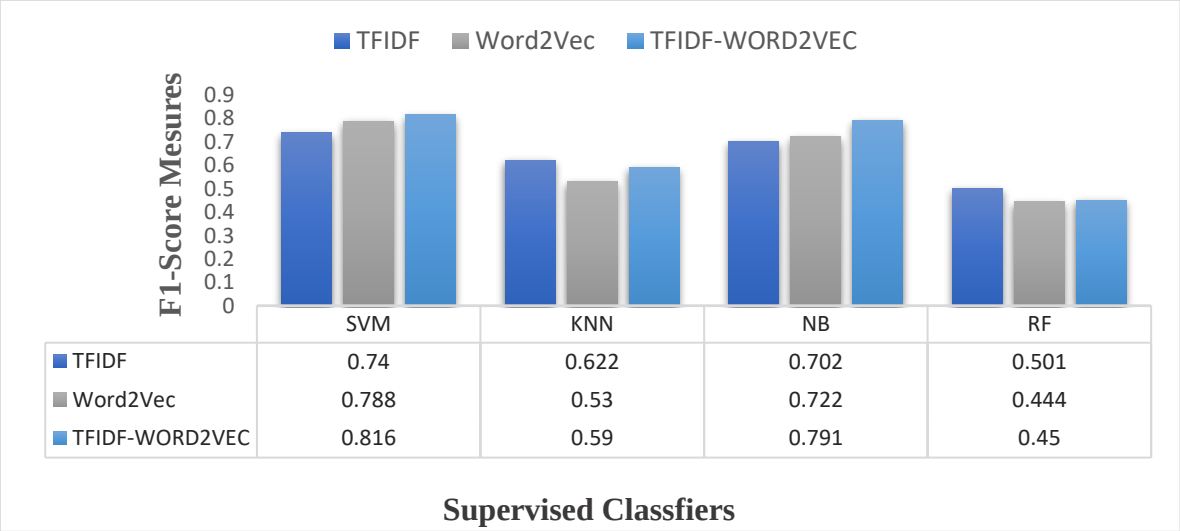


Figure 6.5 Performance of Classifiers using Feature Extractions methods

6.1.1.3.1 Pattern Group Recommendation Results Using SVM

The classification results obtained by SVM via the outperformed feature extraction method (Table 6.2) performed on GoF design problems are depicted in Table 6.6. The correct recommended design pattern class is labeled as bold with underline whereas the incorrect is labeled as bold with a broken line as shown in Table 6.6. Subsequently, the same recommendations are given for rest of design problems in Appendix D.1 in Table D.1.1.

Table 6.6 Recommended Pattern Class Using SVM Model with Outperformed Extraction Method

Design Problem	**Correct Pattern Group**	**Recommended Pattern Group**
Design Problem 1	Behavioral	**Creational**
Design Problem 2	Behavioral	**Behavioral**
Design Problem 3	Behavioral	**Behavioral**
Design Problem 4	Structural	**Structural**
Design Problem 5	Structural	**Structural**
Design Problem 6	Creational	**Creational**
Design Problem 7	Structural	**Structural**
Design Problem 8	Structural	**Behavioral**
Design Problem 9	Creational	**Creational**
Design Problem 10	Structural	**Structural**
Design Problem 11	Creational	**Creational**
Design Problem 12	Creational	**Creational**
Design Problem 13	Structural	**Structural**
Design Problem 14	Behavioral	**Creational**
Design Problem 15	Behavioral	**Behavioral**

Based on Table 6.7, the performance metrics for the SVM model using features by a combination of Word2Vec with the TF-IDF method obtain precision of 81.2%,71.4%,86.7% for Behavioral, Creational, and Structural respectively. The calculated recall with result scores 76.5%,83.3%, and 86.7% for Behavioral, Creational, and Structural respectively. However, the calculated F1-score of the SVM model is 78.8%, 76.9%,86.7% for Behavioral, Creational, Structural respectively.

Table 6.7 Classification Report Result for SVM using Word2vec Weighted by TF-IDF Features

Design Pattern Class	**SVM Model Performance Measures**		
	Precision	***Recall***	***F1-Score***
Behavioral	0.812	0.765	0.788
Creational	0.714	0.833	0.769
Structural	0.867	0.867	0.867
Micro Avg	0.816	0.816	0.816
Macro Avg	0.798	0.822	0.808
Weighted Avg	0.818	0.816	0.816

Figure 6.6 illustrates the sampled confusion matrix of SVM model classification results for the design patterns categories. They are described below for each category

- **Behavioral Class:** The feature extraction method TF-IDF yields the best result among others. 15 design problems corrected classified using TF-IDF, 10 design problems corrected classified using Word2Vec, and 13 design problems corrected classified using Word2Vec weighted by TF-IDF out of 17 respectively, whereas the others misclassified incorrectly.
- **Creational Class:** The combination of Word2Vec with the TF-IDF feature extraction method enabled gives us the best result than the others. 5 design problems corrected classified using TF-IDF, 6 design problems corrected classified using Word2Vec, and 5 design problems corrected classified using Word2Vec weighted by TF-IDF out of 6 respectively, whereas the others misclassified incorrectly. Features from TF-IDF and Word2Vec weighted by TF-IDF have encountered the same classification results.
- **Structural Class:** Word2Vec feature extraction method classifier performs excellent results, while other methods do not work well. 8 design problems corrected classified using TF-IDF, 13 design problems corrected classified using Word2Vec, and 13 design problems corrected classified using Word2Vec weighted by TF-IDF out of 15 respectively, whereas the others misclassified incorrectly. Features from Word2Vec and Word2Vec weighted by TF-IDF have encountered the same classification results.

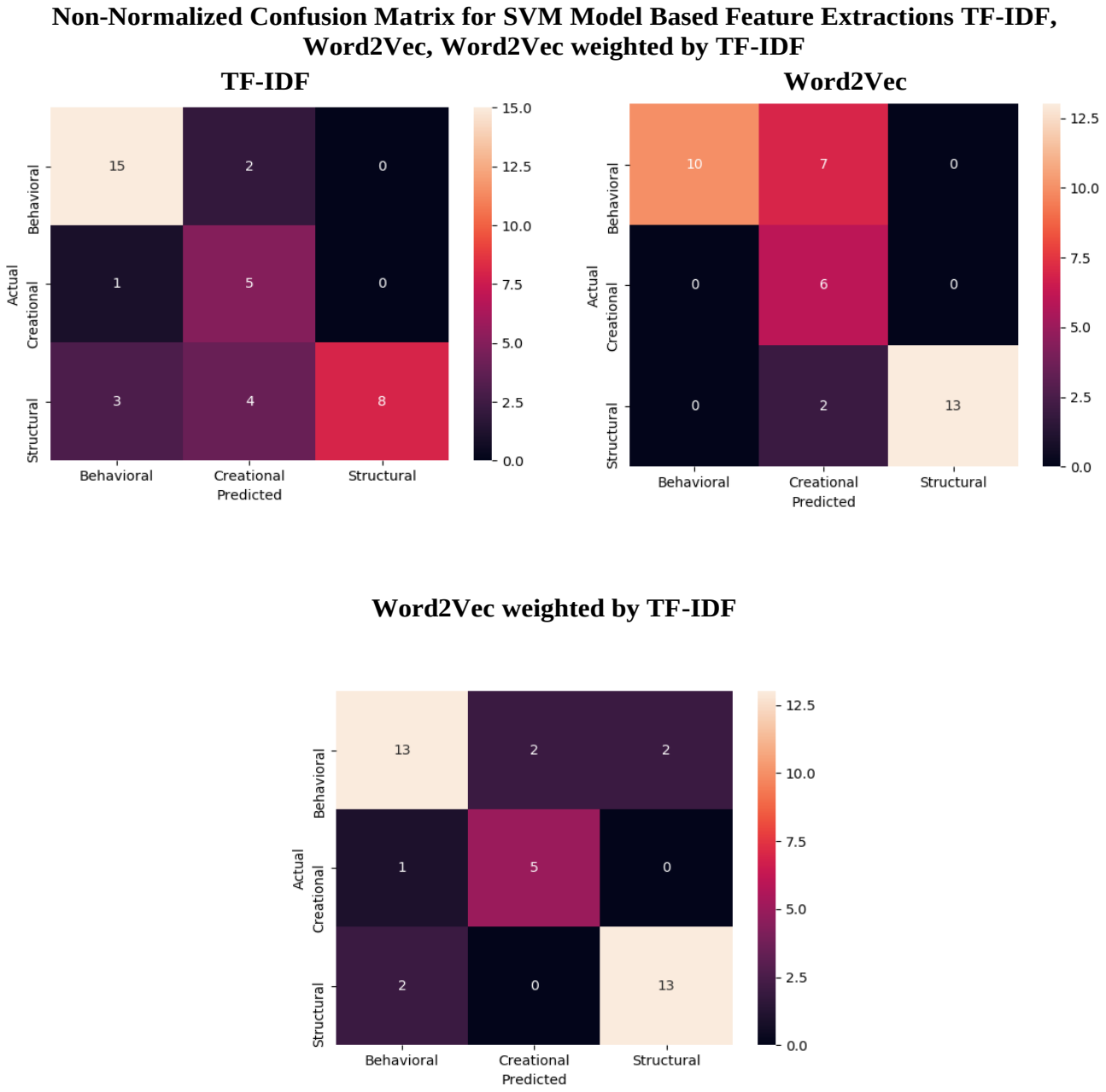


Figure 6.6 Confusion Matrix of SVM Models Based on Extracted Features

6.1.1.3.2 Pattern Group Recommendation Results Using K-NN

The classification results obtained by K-NN via the outperformed feature extraction method (Table 6.3) performed on GoF design problems are depicted in Table 6.8. The correct recommended design pattern class is labeled as bold with underline whereas the incorrect is labeled as bold with a broken line. Subsequently, ten design problems are listed as shown in Table 6.8 and while the same recommendations are given for the rest of design problems in Appendix D.1 in Table D.1.2.

Table 6.8 Recommended Pattern Class Using K-NN Model with Outperformed Extraction Method

Design Problem	**Correct Pattern Group**	**Recommended Pattern Group**
Design Problem 1	Behavioral	**Structural**
Design Problem 2	Behavioral	**Creational**
Design Problem 3	Behavioral	**Creational**
Design Problem 4	Structural	**Creational**
Design Problem 5	Structural	**Structural**
Design Problem 6	Creational	**Creational**
Design Problem 7	Structural	**Structural**
Design Problem 8	Structural	**Structural**
Design Problem 9	Creational	**Structural**
Design Problem 10	Structural	**Structural**
Design Problem 11	Creational	**Creational**
Design Problem 12	Creational	**Creational**
Design Problem 13	Structural	**Structural**
Design Problem 14	Behavioral	**Behavioral**
Design Problem 15	Behavioral	**Creational**

Based on Table 6.9, the performance metrics for the K-NN model using features extracted by the TF-IDF method obtain precision of 87.5%,31.2%,78.6% for Behavioral, Creational, and Structural respectively. The calculated recall with result scores 41.2%,83.3%, and 73.3% for Behavioral, Creational, and Structural respectively. However, the calculated F1-score of the K-NN model is 56.0%, 45.5%,75.9% for Behavioral, Creational, Structural respectively.

Table 6.9 Classification Report for K-NN using TF-IDF Features

Design Pattern Class	**K-NN Model Performance Measures**		
	Precision	***Recall***	***F1-Score***
Behavioral	0.875	0.412	0.560
Creational	0.312	0.833	0.455
Structural	0.786	0.733	0.759
Micro Avg	0.605	0.605	0.605
Macro Avg	0.658	0.659	0.591
Weighted Avg	0.751	0.605	0.622

Figure 6.7 illustrates the sampled confusion matrix of K-NN models model classification results for the design patterns categories. They are described below for each category

- **Behavioral Class:** Word2Vec feature extraction method yields the best result among others. 7 design problems corrected classified using TF-IDF, 7 design problems corrected classified using Word2Vec, and 8 design problems corrected classified using Word2Vec weighted by TF-IDF out of 17 respectively, whereas the others misclassified incorrectly. Features from TF-IDF and Word2Vec have encountered the same classification results.
- **Creational Class:** TF-IDF feature extraction method yields the best result among the rest. 5 design problems corrected classified using TF-IDF, 4 design problems corrected classified using Word2Vec, and 4 design problems corrected classified using Word2Vec weighted by TF-IDF out of 6 respectively, whereas the others misclassified incorrectly. The last two feature extraction methods have the same number classifications. Features from Word2Vec and Word2Vec weighted by TF-IDF have encountered the same classification results.
- **Structural Class:** The combination of Word2Vec with the TF-IDF feature extraction method gives better results than others. 11 design problems corrected classified using TF-IDF, 9 design problems corrected classified using Word2Vec, and all 10 design problems corrected classified using Word2Vec weighted by TF-IDF out of 15 respectively, whereas the others misclassified incorrectly.

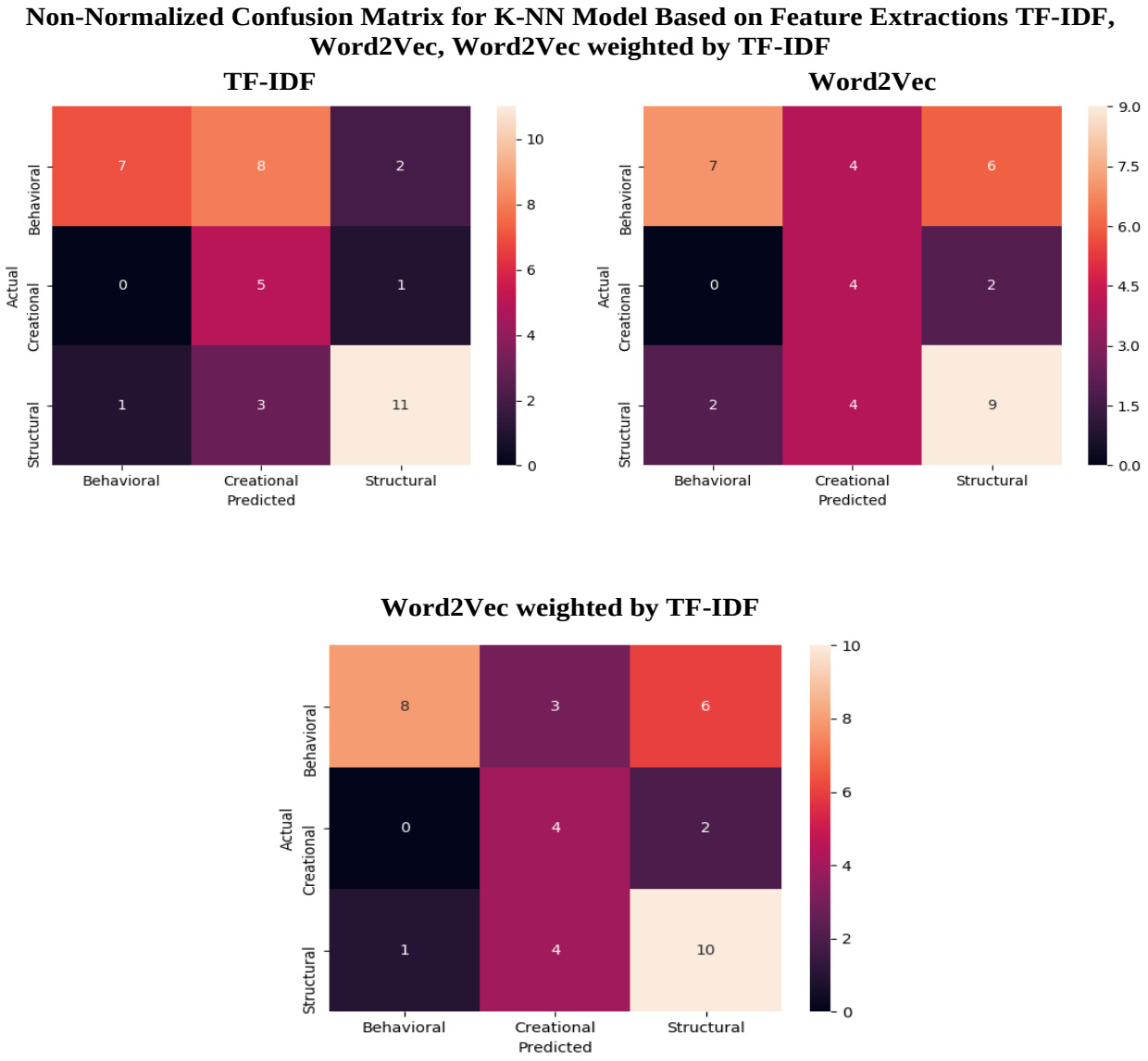


Figure 6.7 Confusion Matrix of K-NN Models Based on Extracted Features

6.1.1.3.3 Pattern Group Recommendation Results Using NB

The classification results obtained by NB via the outperformed feature extraction method (Table 6.4) performed on GoF design problems are depicted in Table 6.10. The correct recommended design pattern class is labeled as bold with underline whereas the incorrect is labeled as bold with a broken line. Subsequently, ten design problems are listed as shown in Table 6.10 and while the same recommendations are given for the rest of design problems in Appendix D.1 in Table D.1.3.

Table 6.10 Recommended Pattern Class Using NB Model with Outperformed Extraction Method

Design Problem	**Correct Pattern Group**	**Recommended Pattern Group**
Design Problem 1	Behavioral	**Behavioral**
Design Problem 2	Behavioral	**Behavioral**
Design Problem 3	Behavioral	**Behavioral**
Design Problem 4	Structural	**Structural**
Design Problem 5	Structural	**Structural**
Design Problem 6	Creational	**Creational**
Design Problem 7	Structural	**Structural**
Design Problem 8	Structural	**Structural**
Design Problem 9	Creational	**Creational**
Design Problem 10	Structural	**Behavioral**
Design Problem 11	Creational	**Creational**
Design Problem 12	Creational	**Creational**
Design Problem 13	Structural	**Structural**
Design Problem 14	Behavioral	**Behavioral**
Design Problem 15	Behavioral	**Behavioral**

Based on Table 6.11, the performance metrics for the NB model using features by a combination of Word2Vec with the TF-IDF method obtain precision of 73.7%,100%,78.6% for Behavioral, Creational, and Structural respectively. The calculated recall with result scores 82.4%,83.3%, and 73.3% for Behavioral, Creational, and Structural respectively. However, the calculated F1-score of the NB model is 77.8%,90.9%,75.9% for Behavioral, Creational, Structural respectively.

Table 6.11 Classification Report for NB using Word2vec Weighted by TF-IDF Features

Design Pattern Class	**NB Model Performance Measures**		
	Precision	***Recall***	***F1-Score***
Behavioral	0.737	0.824	0.778
Creational	1.000	0.833	0.909
Structural	0.786	0.733	0.759
Micro Avg	0.789	0.789	0.789
Macro Avg	0.841	0.797	0.815
Weighted Avg	0.798	0.789	0.791

Figure 6.8 illustrates the sampled confusion matrix of NB models model classification results for the design patterns categories. They are described below for each category

- **Behavioral Class:** The combination of Word2Vec with the TF-IDF feature extraction method gives a better result than others. 15 design problems corrected classified using TF-IDF, 8 design problems corrected classified using Word2Vec, and 14 design problems corrected classified using Word2Vec weighted by TF-IDF out of 17 respectively, whereas the others misclassified incorrectly.
- **Creational Class:** The feature extraction method Word2Vec yields the best result among the rest. 5 design problems corrected classified using TF-IDF, 5 design problems corrected classified using Word2Vec, and 5 design problems corrected classified using Word2Vec weighted by TF-IDF out of 6 respectively, whereas the others misclassified incorrectly. Features from all extraction methods have encountered the same classification results.
- **Structural Class:** The combination of Word2Vec with the TF-IDF feature extraction method gives better results than others. 7 design problems corrected classified using TF-IDF, 15 design problems corrected classified using Word2Vec, and 11 design problems corrected classified using Word2Vec weighted by TF-IDF out of 15 respectively, whereas the others misclassified incorrectly.

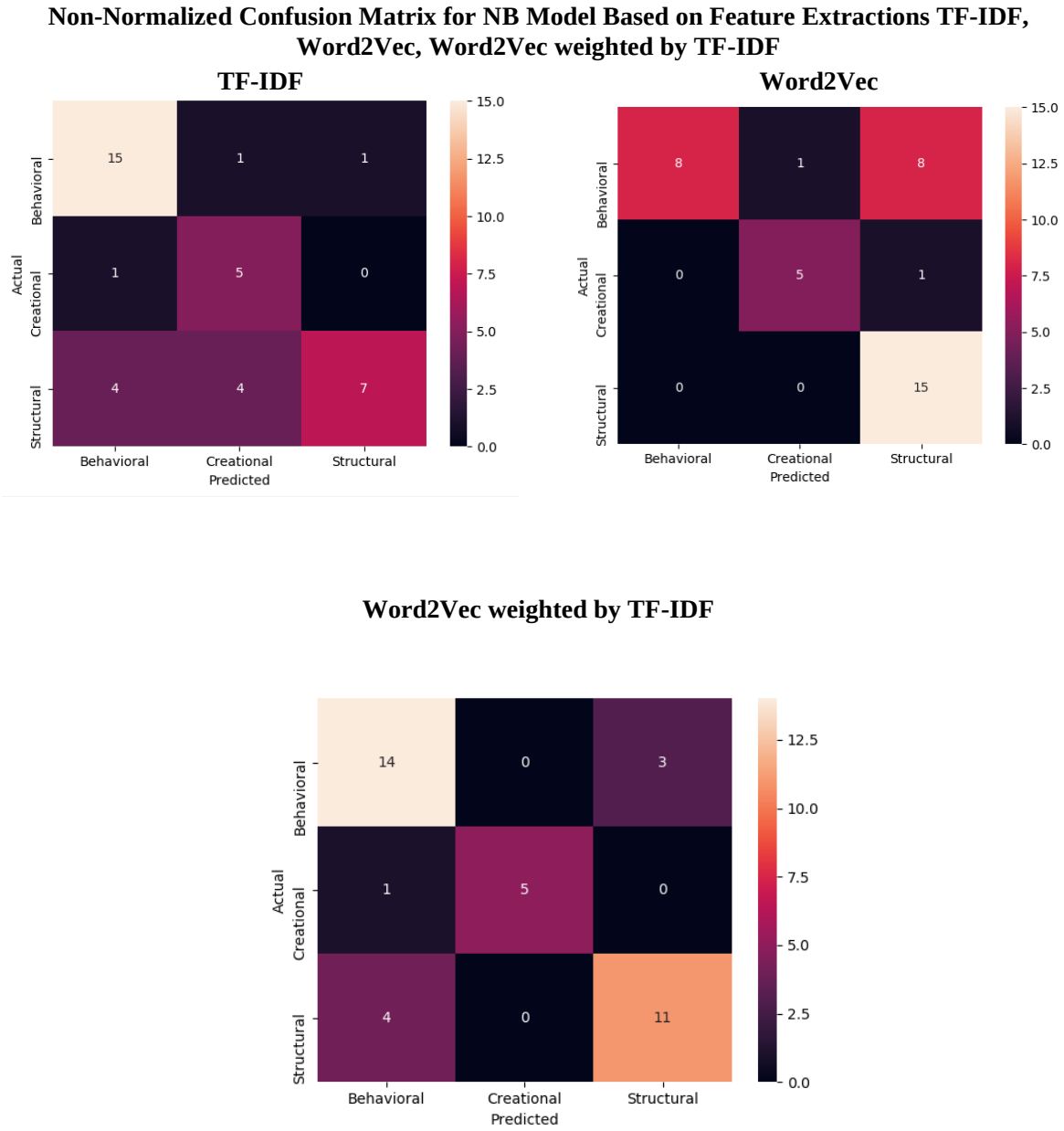


Figure 6.8 Confusion Matrix of NB Models Based on Extracted Features

6.1.1.3.4 Pattern Group Recommendation Results Using RF

The classification results obtained by RF via the outperformed feature extraction method (Table 6.5) performed on GoF design problems are depicted in Table 6.12. The correct recommended design pattern class is labeled as bold with underline whereas the incorrect is labeled as bold with a broken line. Subsequently, ten design problems are listed as shown in Table 6.12 and while the same recommendations are given for the rest of design problems in Appendix D.1 in Table D.1.4.

Table 6.12 Recommended Pattern Class Using RF Model with Outperform Extraction Method

Design Problem	**Correct Pattern Group**	**Recommended Pattern Group**
Design Problem 1	Behavioral	**Structural**
Design Problem 2	Behavioral	**Creational**
Design Problem 3	Behavioral	**Creational**
Design Problem 4	Structural	**Creational**
Design Problem 5	Structural	**Structural**
Design Problem 6	Creational	**Creational**
Design Problem 7	Structural	**Structural**
Design Problem 8	Structural	**Behavioral**
Design Problem 9	Creational	**Structural**
Design Problem 10	Structural	**Structural**
Design Problem 11	Creational	**Creational**
Design Problem 12	Creational	**Creational**
Design Problem 13	Structural	**Structural**
Design Problem 14	Behavioral	**Creational**
Design Problem 15	Behavioral	**Creational**

Based on Table 6.13, the performance metrics for the RF model using features extracted by the TF-IDF method obtain precision of 66.7%,26.3%,76.9% for Behavioral, Creational, and Structural respectively. The calculated recall with result scores 23.5%,83.3%, and 66.7% for Behavioral, Creational, and Structural respectively. However, the calculated F1-score of the RF model is 34.8%, 40.0%,71.4% for Behavioral, Creational, Structural respectively.

Table 6.13 Classification Report for RF using TF-IDF Features

Design Pattern Class	**RF Model Performance Measures**		
	Precision	***Recall***	***F1-Score***
Behavioral	0.667	0.235	0.348
Creational	0.263	0.833	0.400
Structural	0.769	0.667	0.714
Micro Avg	0.500	0.500	0.500
Macro Avg	0.566	0.578	0.487
Weighted Avg	0.643	0.500	0.501

Figure 6.9 illustrates the sampled confusion matrix of NB models model classification results for the design patterns categories. They are described below for each category

- **Behavioral Class:** TF-IDF feature extraction method gives a better result than others. 4 design problems corrected classified using TF-IDF, 2 design problems corrected classified using Word2Vec, and 2 design problems corrected classified using Word2Vec weighted by TF-IDF out of 17 respectively, whereas the others misclassified incorrectly. Features from Word2Vec and Word2Vec weighted by TF-IDF have encountered the same classification results.
- **Creational Class:** Word2Vec feature extraction method yields the best result among the rest. 5 design problems corrected classified using TF-IDF, 6 design problems corrected classified using Word2Vec, and 6 design problems corrected classified using Word2Vec weighted by TF-IDF out of 6 respectively, whereas the others misclassified incorrectly. Features from Word2Vec and Word2Vec weighted by TF-IDF have encountered the same classification results.
- **Structural Class:** Word2Vec and combination of Word2Vec with TF-IDF feature extraction methods gives a better result than TF-IDF. 10 design problems corrected classified using TF-IDF, 10 design problems corrected classified using Word2Vec, and 10 design problems corrected classified using Word2Vec weighted by TF-IDF out of 15 respectively, whereas the others misclassified incorrectly. Features from all extraction methods have encountered the same classification results.

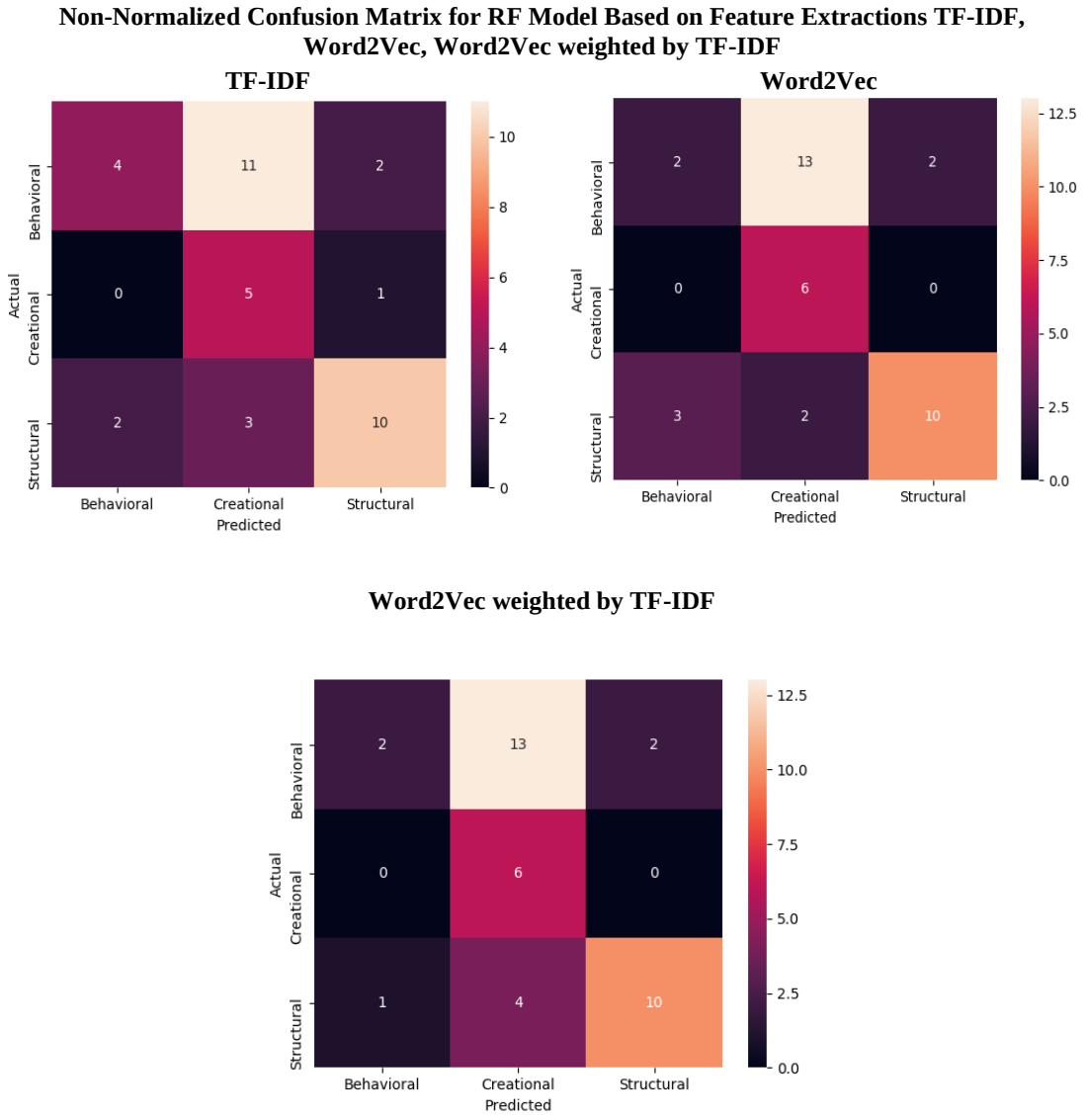


Figure 6.9 Confusion Matrix of RF Models Based on Extracted Features

6.1.2 Recommendation of Design Patterns from Suggested Design Pattern Group

In the second phase, i.e., design pattern retrieval, from design patterns classes obtained from the previous phase, a recommended design pattern class that is similar to a given design problem is determined based on cosine similarity. In this regard, we have calculated the cosine similarity between Word2Vec-TFIDF vectors of the actual design pattern with each design pattern of the recommend group using the implementation in section 5.6.

Table 6.14 illustrates the sample recommended design pattern with maximum cosine similarity value is selected followed by the other two design patterns. The first recommended design pattern labeled as bold and the incorrect recommended labeled as bold Italic broken

line. The purpose of listing the other two recommended design patterns is to narrow down the selection process for the designer if the first recommended pattern is wrongly suggested.

We have provided the cosine similarity results for all recommended software design pattern groups in Appendix D.2. The cosine similarity results show that out of 31 design problems with the right recommended design pattern group, an incorrect design pattern was recommended for 7. Even though 7 design problems had incorrect recommendations, 2 of them had their correct design pattern not recommended on the top three list.

From the similarity experiment result, we observed Cosine Euclidean Distance computation between the pattern vectors show that the design problem scenarios with more words have a higher probability of selecting the right design pattern. For instance, Design Problems with ID 4,9,10,12,15,17 and 26 have fewer words than others in their problem scenarios.

Table 6.14 Cosine Similarity Results for 10 Sample Design Problems

Design Problem ID	**Recommended DP Group**	**Correct Pattern**	**1^{st} Selected Pattern**	$\boldsymbol{CS_1}$	**2^{nd} Selected pattern**	$\boldsymbol{CS_2}$	**3^{rd} Selected pattern**	$\boldsymbol{CS_3}$
2	**Behavioral**	Interpreter	**Interpreter**	**0.370**	Memento	0.061	Strategy	0.021
3	**Behavioral**	Iterator	**Iterator**	**0.211**	Interpreter	0.093	Memento	0.039
4	**Structural**	Facade	***Proxy***	**0.187**	Facade	0.181	Adapter	0.161
5	**Structural**	proxy	**proxy**	**0.302**	Bridge	0.161	Composite	0.071
6	**Creational**	Singleton	**Singleton**	**0.432**	Abstract Factory	0.303	Factory Method	0.111
7	**Structural**	Composite	**Composite**	**0.125**	Flyweight	0.052	Facade	0.022

9	**Creational**	Singleton	***Builder***	**0.085**	Prototype	0.019	Singleton	0.015
10	**Structural**	Proxy	***Facade***	**0.098**	Proxy	0.032	Decorator	0.028
11	**Creational**	Abstract Factory	**Abstract Factory**	**0.212**	Prototype	0.154	Singleton	0.178
12	**Creational**	Factory Method	***Singleton***	**0.235**	Builder	0.017	Factory Method	0.015

According to the precision values of the best-supervised classifiers that are given in Tables 6.8, 6.9, 6.11, and 6.13., we observe that every classifier has a precision and the recall value is less than 1 or equal to 1 for each specific category. The meaning of precision 1 is, if a classifier predicts an instance belongs to a particular design pattern class, it indeed belongs to that pattern class. However, if a recall is less than 1. This means there may be some chances that classifiers might predict incorrectly in a case where an instance belongs to the pattern class. Therefore, the manual software design selection is still required for a design problem that is predicted incorrectly by every supervised classifier as shown on the above confusion matrix diagrams.

Under this study of experiments, we observed the significant impact of the feature set constructed through each feature extraction method on the performance of classifiers and also on the cosine similarity of the vectors.

Feature extraction methods are key processes to capture patterns from sample dataset to models using a supervised machine learning algorithm. The result of this study shows that features from word2vec weighted by TF-IDF for SVM, NB, RF perform better accuracy than both feature extraction methods due to the abilities to bring extra semantic features weighted by TF-IDF words that result in a better score.

This claim is also shared by some of the other research papers [63] that used word2vec with weighted TF-IDF as a feature extraction method. Besides the fact that the ensemble of Word2Vec and TF-IDF feature extraction methods enhance the performance of SVM, the ability of the classifier to handle an imbalance distribution of class in the dataset [79], is also one of the reasons for a major significance in achieving overall better performance among the other three classifiers.

CHAPTER SEVEN

7 Conclusion, Recommendation and Future Work

In this chapter, the proposed method of software design pattern recommending using text classification and the finding of this study thesis is concluded. It also describes the recommendation and future research directions.

7.1 Conclusion

This research proposed to recommend software design patterns for a given software design problem using machine learning text classification algorithms. The study attempted to develop text classifiers and evaluated for finding the best feature extraction methods and a suitable training set. This can be used for recommending a software design pattern group based on text relevancy of design problem descriptions. Text classification provides a bridge connecting group recommendation with an automatic process.

The experimental results illustrate that the proposed method based on the text classification approach via supervised learners aid in giving a promising base for a recommendation of design pattern group and using cosine similarity to recommend more right design pattern(s) for a particular design problem. The proposed method aims to construct a more illustrative feature set to improve the performance of supervised classifiers employed in the text classification based automated system, specifically in software design pattern automation. The proposed method has two phases, pattern group recommendation, and suggestion of design Pattern(s) from that group. To implement the proposed method, we undertake a case study on the GoF design pattern collection by constructing training and test dataset from the problem description of software design pattern and real-design problem scenario descriptions respectively.

The SVM, NB, K-NN, and RF supervised classifiers are used in the proposed method. Numerous performance measures are used to evaluate the quality of text classification. We also develop an evaluation model by leveraging the well-known measures to evaluate the effectiveness of the proposed method. Moreover, in the proposed method we used text preprocessing techniques followed by features extraction methods such as TF-IDF, Word2Vec, and Word2Vec weighted by TF-IDF.

We observed a significant variation in the performance of supervised learning classifiers with the feature extraction methods. NB performs a 100% precision score with correct classification for the creational design pattern class. RF with all feature extraction methods does not perform well as compared to other supervised classifiers of the proposed method. However, the SVM classifier with all feature extraction methods outperforms better than others supervised classifiers. SVM with Word2Vec weighted by the TF-IDF feature extraction method obtained 81.6% high weighted average F1-score. Therefore, the SVM classifier can be recommended to exploit the proposed method. Subsequently, in terms of determining the best feature extraction methods, we observed the combination of TF-IDF and Word2Vec performs better than TF-IDF and Word2Vec.

Finally, cosine similarity is used to calculate the similarity score to suggest the appropriate design patterns to software designers or developers.

The importance of this comparative experimental study is to construct semantical illustrative features using ensemble feature extraction methods of TF-IDF and Word2Vec, that capture and collect as much information as possible about all the software design problem descriptions. The ultimate aim of the study is, however, to enrich the recommendation process of software design pattern with more features, which have been found to improve its precision.

7.2 Recommendations

The proposed methods from this study motivate the novice designer on the recommendation of design pattern(s) for a design problem at hand. Thus, the software practitioner of any domain needs to adopt this study to improve the performance of text classification approach in the software design industry.

7.3 Future works

The proposed method can be improved by increasing the comparison of different methods and the feature vector size while the reduction of sparsity terms also has to be considered. For a better result, it would be better to apply the study on more software design pattern collections to analyze the variation of problems and recommendations of design patterns.

Although the experiment is performed on an imbalance training dataset, the proposed method can be enhanced and well-justified by considering the SMOTE technique to balance the dataset.

The study performance is based on a single test dataset that may not reflect the consistent accuracy when our classifiers are used with new test dataset. Therefore, to produce reasonable predictions of how well the model will perform on future design problem samples, cross-validation techniques should be utilized with large design pattern datasets. The study should also employ other machine learning algorithms such as unsupervised techniques. Unsupervised techniques clusters design patterns of any target catalog, which are recorded using either the same or different classification schemes that can aid developers to organize the design based on their text relevancy.

References

[1] W. G. Wood, “A Practical Example of Applying Attribute-Driven Design (ADD), Version 2.0 Software Architecture Technology Initiative,” 2007.

[2] C. Alexander, S. Ishikawa, and M. Silverstein, “A Pattern Language: Towns, Buildings, Construction (Center for Environmental Structure Series),” *Book*, 1978.

[3] L. Rising, “Design patterns in communications software,” *IEEE Commun. Mag.*, vol. 37, no. 4, pp. 32–33, 1999, doi: 10.1109/MCOM.1999.755445.

[4] “Almanac 2000.” [Online]. Available: http://www.smallmemory.com/almanac/.

[5] “EuroPLoP.” [Online]. Available: http://europlop.net/content/repositories.

[6] “Portland Pattern Repository.” [Online]. Available: http://c2.com/ppr/titles.html.

[7] A. Birukou, “A survey of existing approaches for pattern search and selection,” p. 1, 2012, doi: 10.1145/2328909.2328912.

[8] I. Sommerville, *Software engineering*, 7th Editio. Boston, MA, USA: Addison-Wesley, 2004.

[9] C. Bouhours, H. Leblanc, and C. Percebois, “Spoiled patterns: how to extend the GoF,” *Softw. Qual. J.*, vol. 23, no. 4, pp. 661–694, 2015, doi: 10.1007/s11219-014-9249-z.

[10] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, “Design Patterns: Elements of Reusable Software,” p. 395, 1996.

[11] W. Pree and H. Sikora, “Design patterns for object-oriented software development (tutorial),” pp. 663–664, 2004, doi: 10.1145/253228.253810.

[12] M. Doernhoefer, “Book reviews: Object models strategies, patterns, and applications, Peter coad,” Softw. Eng. Notes, vol. 23, no. 3, pp. 125–125, 1998

[13] B. P. Douglass, “Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems,” p. 528, 2002.

[14] I. Idris and A. Selamat, “Improved email spam detection model with negative selection algorithm and particle swarm optimization,” *Appl. Soft Comput. J.*, vol. 22, pp. 11–27, 2014, doi: 10.1016/j.asoc.2014.05.002.

[15] S. Hussain *et al.*, “Implications of deep learning for the automation of design patterns organization,” *J. Parallel Distrib. Comput.*, vol. 117, pp. 256–266, 2018, doi: 10.1016/j.jpdc.2017.06.022.

[16] W. Medhat, A. Hassan, and H. Korashy, “Sentiment analysis algorithms and applications: A survey,” *Ain Shams Eng. J.*, vol. 5, no. 4, pp. 1093–1113, 2014, doi: 10.1016/j.asej.2014.04.011.

[17] C. Zhang, X. Wu, Z. Niu, and W. Ding, “Authorship identification from unstructured texts,” *Knowledge-Based Syst.*, vol. 66, pp. 99–111, 2014, doi: 10.1016/j.knosys.2014.04.025.

[18] S. Hussain, J. Keung, A. A. Khan, and K. E. Bennin, “A Methodology to Automate

the Selection of Design Patterns," *Proc. - Int. Comput. Softw. Appl. Conf.*, vol. 2, pp. 161–166, 2016, doi: 10.1109/COMPSAC.2016.226.

[19] S. Hussain, J. Keung, and A. A. Khan, "Software design patterns classification and selection using text categorization approach," *Appl. Soft Comput. J.*, vol. 58, pp. 225–244, 2017, doi: 10.1016/j.asoc.2017.04.043.

[20] I. Ali, M. Asif, M. Shahbaz, A. Khalid, M. Rehman, and A. Guergachi, "Text categorization approach for secure design pattern selection using software requirement specification," *IEEE Access*, vol. 6, pp. 73928–73939, 2018, doi: 10.1109/ACCESS.2018.2883077.

[21] W. Rui, J. Liu, and Y. Jia, "Unsupervised feature selection for text classification via word embedding," *Proc. 2016 IEEE Int. Conf. Big Data Anal. ICBDA 2016*, 2016, doi: 10.1109/ICBDA.2016.7509787.

[22] A. Shvets, Design Pattern Explained Simply, 1st ed. Sourcemaking.com, 2010

[23] J. R. T. Alan Shalloway, "Design Patterns Explained: A New Perspective on Object-Oriented Design, Second Edition," 2004.

[24] W. G. Wood, "A Practical Example of Applying Attribute-Driven Design (ADD), Version 2 . 0," *Technology*, vol. Version 2, no. February, p. 59, 2007.

[25] L. Bass, P. Clements, and R. Kazman, "Software Architecture in Practice (3rd Edition)," *Architecture*, p. 528, 2012.

[26] D. K. Van Duyne, J. A. Landay, and J. I. Hong, "The design of sites : patterns, principles, and processes for crafting a customer-centered Web experience," p. 762, 2003.

[27] J. Tidwell, Designing Interfaces: Patterns for Effective Interaction Design, 2nd ed. O'Reilly Media, 2010.

[28] M. Schumacher, E. Fernandez-Buglioni, D. Hybertson, F. Buschmann, and P. Sommerlad, "Security Patterns: Integrating Security and Systems Engineering," *Syst. Eng.*, p. 600, 2006.

[29] H. Baraki *et al.*, "Interdisciplinary Design Patterns for Socially Aware Computing," *Proc. - Int. Conf. Softw. Eng.*, vol. 2, pp. 477–486, 2015, doi: 10.1109/ICSE.2015.180.

[30] W. F. Tichy, "Catalogue of general-purpose software design patterns," *Proc. Conf. Technol. Object-Oriented Lang. Syst. TOOLS*, pp. 330–339, 1997, doi: 10.1109/tools.1997.654742.

[31] W. Zimmer, "Relationships between Design Patterns," *Design*, vol. 1, pp. 345–364, 1995, doi: 10.1.1.37.8779.

[32] S. M. H. Hasheminejad and S. Jalili, "Design patterns selection: An automatic two-phase method," *J. Syst. Softw.*, vol. 85, no. 2, pp. 408–424, 2012, doi: 10.1016/j.jss.2011.08.031.

[33] P. Velasco-Elizondo, R. Marín-Piña, S. Vazquez-Reyes, A. Mora-Soto, and J. Mejia, "Knowledge representation and information extraction for analysing architectural patterns," *Sci. Comput. Program.*, vol. 121, pp. 176–189, 2016, doi: 10.1016/j.scico.2015.12.007.

[34] D. K. Kim and C. El Khawand, “An approach to precisely specifying the problem domain of design patterns,” *J. Vis. Lang. Comput.*, vol. 18, no. 6, pp. 560–591, 2007, doi: 10.1016/j.jvlc.2007.02.009.

[35] N.L. Hsueh, “Object-Oriented design: a goal-driven and pattern-based approach,” pp. 1–18, 2007.

[36] S. Hasso and C. R. Carlson, “A Theoretically-based Process for Organizing Design Patterns,” *12th Pattern Lang. Programs, (PLoP 2005)*, pp. 1–22, 2005.

[37] P. El Khoury, A. Mokhtari, E. Coquery, and M. S. Hacid, “An ontological interface for software developers to select security patterns,” *Proc. - Int. Work. Database Expert Syst. Appl. DEXA*, pp. 297–301, 2008, doi: 10.1109/DEXA.2008.110.

[38] E. Blomqvist, “Pattern ranking for semi-automatic ontology construction,” *Proc. ACM Symp. Appl. Comput.*, pp. 2248–2255, 2008, doi: 10.1145/1363686.1364224.

[39] S. Henninger and V. Corrêa, “Software Pattern Communities: Current Practices and Challenges,” *Proc. 14th Conf. Pattern Lang. Programs (PLOP ’07), Monticello (IL), USA, 5-8 Sept. 2007*, pp. 14:1--14:19, 2007, doi: 10.1145/1772070.1772087.

[40] P. Gomes *et al.*, “Using CBR for automation of software design patterns,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 2416, pp. 534–548, 2002, doi: 10.1007/3-540-46119-1_39.

[41] N. Sanyawong and E. Nantajeewarawat, “Design pattern recommendation based-on a pattern usage hierarchy,” *2014 Int. Comput. Sci. Eng. Conf. ICSEC 2014*, pp. 134–139, 2014, doi: 10.1109/ICSEC.2014.6978183.

[42] A. K. Uysal, “An improved global feature selection scheme for text classification,” *Expert Syst. with Appl.*, 2016.

[43] and S.-S. H. Q. Wu, Y. Ye, H. Zhang, M. K. Ng, “FORESTEXTER : An efficient random forest algorithm for imbalanced text categorization,” *Knowledge-Based Syst.*, pp. 105–116, 2014.

[44] and A. E. H. T-H. Chen, S. W. Thomas, “A survey on the use of topic models when mining software repositories,” *Empir. Softw. Eng.*, 2016.

[45] C. Fox, “Lexical analysis and stoplists,” *Data Struct. Algorithms*, pp. 102–130, 1992.

[46] C. Kluckhohn, : “Human Behavior and the Principle of Least Effort . George Kingsley Zipf.,” *Am. Anthropol.*, vol. 52, no. 2, pp. 268–270, 1950, doi: 10.1525/aa.1950.52.2.02a00290.

[47] R. Tsz-Wai Lo, B. He, and I. Ounis, “Automatically Building a Stopword List for an Information Retrieval System,” 2005.

[48] T. M. Cover and J. A. Thomas, “Elements of Information Theory,” *Elem. Inf. Theory*, vol. 2nd Editio, pp. 1–748, 2006, doi: 10.1002/047174882X.

[49] M. F. Porter, “An algorithm for suffix stripping,” *Program*, vol. 40, no. 3, pp. 211–218, 2006, doi: 10.1108/00330330610681286.

[50] C. D. Paice, “Another Stemmer,” *ACM SIGIR Forum*, vol. 24, no. 3, pp. 56–61, 1990, doi: 10.1145/101306.101310.

[51] J. B. Lovins, “Development of a stemming algorithm,” *Mech. Transl. Comput. Linguist.*, vol. 11, no. June, pp. 22–31, 1968.

[52] M. Melucci and N. Orio, “A novel method for stemmer generation based on Hidden Markov Models,” *Int. Conf. Inf. Knowl. Manag. Proc.*, pp. 131–138, 2003, doi: 10.1145/956888.956889.

[53] P. Majumder, M. Mitra, S. K. Parui, G. Kole, P. Mitra, and K. Datta, “YASS: Yet another suffix stripper,” *ACM Trans. Inf. Syst.*, vol. 25, no. 4, 2007, doi: 10.1145/1281485.1281489.

[54] G. W. Adamson and J. Boreham, “The use of an association measure based on character structure to identify semantically related pairs of words and document titles,” *Inf. Storage Retr.*, vol. 10, no. 7–8, pp. 253–260, 1974, doi: 10.1016/0020-0271(74)90020-5.

[55] J. H. Paik, M. Mitra, S. K. Parui, and K. Järvelin, “GRAS: An effective and efficient stemming algorithm for information retrieval,” *ACM Trans. Inf. Syst.*, vol. 29, no. 4, 2011, doi: 10.1145/2037661.2037664.

[56] P. Han, S. Shen, D. Wang, and Y. Liu, “The influence of word normalization in English document clustering,” *CSAE 2012 - Proceedings, 2012 IEEE Int. Conf. Comput. Sci. Autom. Eng.*, vol. 2, pp. 116–120, 2012, doi: 10.1109/CSAE.2012.6272740.

[57] Y. Yang and J. O. Pedersen, “A comparative study on feature selection in text categorization,” *Proceeding ICML ’97 Proc. Fourteenth Int. Conf. Mach. Learn.*, pp. 412–420, 1997, doi: 10.1093/bioinformatics/bth267.

[58] C. Lee and G. G. Lee, “Information gain and divergence-based feature selection for machine learning-based text categorization,” *Inf. Process. Manag.*, vol. 42, no. 1 SPEC. ISS, pp. 155–165, 2006, doi: 10.1016/j.ipm.2004.08.006.

[59] G. Forman, “An extensive empirical study of feature selection metrics for text classification,” *J. Mach. Learn. Res.*, vol. 3, pp. 1289–1305, 2003.

[60] Ş. Taşcı and Tunga Güngör, “Comparison of text feature selection policies and using an adaptive framework,” *Expert Syst. Appl.*, vol. 40, pp. 4871–4886, 2013.

[61] S. Günal, “Hybrid feature selection for text classification,” *Turkish J. Electr. Eng. Comput. Sci.*, vol. 20, no. SUPPL.2, pp. 1296–1311, 2012, doi: 10.3906/elk-1101-1064.

[62] L. P. Jing, H. K. Huang, and H. B. Shi, “Improved feature selection approach TFIDF in text mining,” *Proc. 2002 Int. Conf. Mach. Learn. Cybern.*, vol. 2, pp. 944–946, 2002, doi: 10.1109/icmlc.2002.1174522.

[63] S. Hussain, J. Keung, and A. A. Khan, “A framework for ranking of software design patterns,” *Adv. Intell. Syst. Comput.*, vol. 611, pp. 205–215, 2018, doi: 10.1007/978-3-319-61566-0_20.

[64] J. Lilleberg, Y. Zhu, and Y. Zhang, “Support vector machines and Word2vec for text classification with semantic features,” *Proc. 2015 IEEE 14th Int. Conf. Cogn. Informatics Cogn. Comput. ICCI*CC 2015*, pp. 136–140, 2015, doi: 10.1109/ICCI-CC.2015.7259377.

[65] M. Fernández-Delgado, E. Cernadas, S. Barro, and D. Amorim, “Do we need hundreds of classifiers to solve real world classification problems?,” *J. Mach. Learn. Res.*, vol. 15, pp. 3133–3181, 2014, doi: 10.1117/1.JRS.11.015020.

[66] N. Sanyawong and E. Nantajeewarawat, “Design Pattern Recommendation: A Text Classification Approach,” *2015 6th Int. Conf. Inf. Commun. Technol. Embed. Syst. IC-ICTES 2015*, 2015, doi: 10.1109/ICTEmSys.2015.7110810.

[67] A. Bouaziz, C. Dartigues-Pallez, C. Da Costa Pereira, F. Precioso, and P. Lloret, “Short text classification using semantic random forest,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 8646 LNCS, pp. 288–299, 2014, doi: 10.1007/978-3-319-10160-6_26.

[68] L. Jiang, Z. Cai, H. Zhang, and D. Wang, “Naive Bayes text classifiers: A locally weighted learning approach,” *J. Exp. Theor. Artif. Intell.*, vol. 25, no. 2, pp. 273–286, 2013, doi: 10.1080/0952813X.2012.721010.

[69] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, “Classification and regression trees,” *Classif. Regres. Trees*, pp. 1–358, 2017, doi: 10.1201/9781315139470.

[70] B. Trstenjak, S. Mikac, and D. Donko, “KNN with TF-IDF based framework for text categorization,” *Procedia Eng.*, vol. 69, pp. 1356–1364, 2014, doi: 10.1016/j.proeng.2014.03.129.

[71] A. Hamdy and M. Elsayed, “Automatic Recommendation of Software Design Patterns: Text Retrieval Approach,” *J. Softw.*, vol. 13, no. 4, pp. 260–268, 2018, doi: 10.17706/jsw.13.4.260-268.

[72] M. E. ABEER HAMDY, “TOWARDS MORE ACCURATE AUTOMATIC RECOMMENDATION OF SOFTWARE DESIGN PATTERNS,” 2018.

[73] L. Feng, Y. K. Chiam, and S. K. Lo, “Text-Mining Techniques and Tools for Systematic Literature Reviews: A Systematic Literature Review,” *Proc. - Asia-Pacific Softw. Eng. Conf. APSEC*, vol. 2017-Decem, pp. 41–50, 2018, doi: 10.1109/APSEC.2017.10.

[74] M. Hossin and N. Sulaiman, “A Review on Evaluation Metrics For Data Classification Evaluations,” *Int. J. Data Min. Knowl. Manag. Process*, vol. 5, no. 2, pp. 1–11, 2015.

[75] J. Novakovic, A. Veljovi, S. Iiic, Z. Papic, and M. Tomovic, “Evaluation of Classification Models in Machine Learning,” *Theory Appl. Math. Comput. Sci.*, vol. 7, no. 1, pp. 39–46, 2017.

[76] A. Huang, “Similarity measures for text document clustering,” *Proc. New Zeal. Comput. Sci. Res. Student Conf.*, 2008.

[77] R. T. A. Shalloway, “Design Pattern Explained: A New Perspective on Object Oriented Design,” *Addison Wesley*, 2001.

[78] K. Veropoulos, C. Campbell, N. Cristianini, and Others, “Controlling the sensitivity of support vector machines,” *Proc. Int. Jt. Conf. Artif. Intell.*, pp. 55–60, 1999, doi: 10.1.1.42.7895.

Appendixes

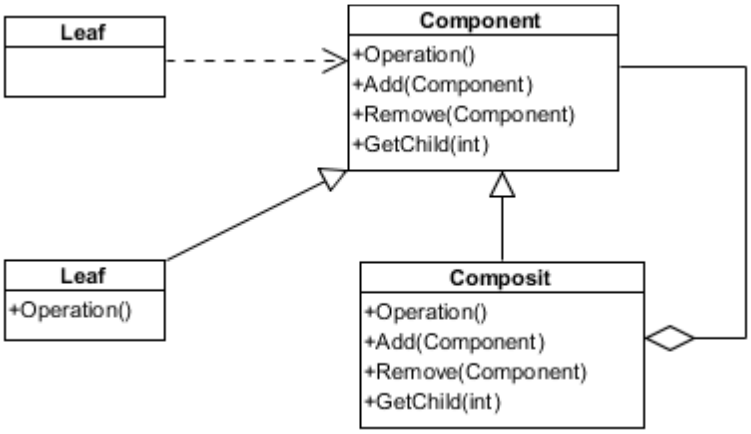
Appendix A: Template of Design Patterns

Table A:1 GoF Design Pattern Template (Composite Design Pattern as an Example)

Design Pattern Name	**Composite**
	Problem Domain
Intent	"Compose objects into tree structures to represent part-whole hierarchies. Composite lets the client treat individual objects and composition of objects uniformly".
Motivation	"Graphics applications like drawing editors and schematic capture systems let users build complex diagrams out of simple components. The user can group components to form larger components, which in turn can be grouped to form still larger components. A simple implementation could define classes for graphical primitives such as Text and Lines plus other classes that act as containers for these primitives. However, there is a problem with this approach: Code that uses these classes must treat primitives and container objects differently, even if most of the time, they use treats them identically. Having to distinguish these objects makes the application more complex. The Composite pattern describes how to use recursive composition so that clients don't have to make this distinction".
Applicability	"The composite pattern applies when there is a part-whole hierarchy of objects, and a client needs to deal with objects uniformly even though an object might be a leaf or a branch."

Solution Domain

Structure



Participant Component, Leaf, Composite, and Client.

Collaborations

Consequences The Composite Pattern

"Defines class hierarchies are consisting of primitive objects and composite objects. Primitive objects can be composed and so on recursively. Whatever client code expects a primitive object, it can also take composite object".

"Makes the client simple. The client can treat composite structure and

"Makes it easier to ass new kind of components. Newly defined Composite or leaf subclasses work automatically with existing structure and client code. Clients do not have to be changed for new Component classes".

"Can make your overall design general. The disadvantage of making it easy to add new components is that it makes it harder to restrict the components of a composite. Sometimes you want a composite to have only certain components. With Composite, you can't rely on the type system to enforce those constraints for you. You will have to use run-time checks instead"

Implementation "The implementation of Composite Design Pattern using C++ constructs".

Related Patterns "Decorator Pattern: When decorators and composites are used together, they will usually have a common parent class."

Appendix B: List and Description of Design Problems

Design Problem 1. (Mapping the relationship between request and handler)**:** "There is a potential variable number of "handler" or "processing element" or "node" objects, and a stream of requests that must be handled. Need to efficiently process the requests without hard-wiring handler relationships and precedence, or request-to-handler mappings."

Design pattern Category: *Behavioral*

Pattern Name: *Chain of Responsibility*

Design Problem 2. (Musician problem of control the pitch of sound)

"The design problem must be defined as grammatical representation for a language and an interpreter to interpret the grammar. Musicians are the main players in the center of the software design. The pitch of a sound and its duration can be represented in musical notation on a staff. This notation provides the language of music. Musicians playing the music from the score are able to reproduce the original pitch and duration of each sound represented."

Design pattern Category: *Behavioral*

Pattern Name: *Interpreter*

Design Problem 3. (To manage the filing system)

"The design problem needs to be addressed to provides ways to access elements of an aggregate object sequentially without exposing the underlying structure of the object. Files are aggregate objects. In office settings where access to files is made through the administrative or secretarial staff, the Iterator pattern is demonstrated with the Secretary acting as the Iterator. Several television comedy skits have been developed around the premise of an executive trying to understand the secretary's filing system. To the executive, the filing system is confusing and illogical, but the secretary is able to access files quickly and efficiently. "

Design pattern Category: *Behavioral*

Pattern Name: *Iterator*

Design Problem 4. (To provide a simplified interface for clients)

"A segment of the client community needs a simplified interface to the overall functionality of a complex subsystem. "

Design pattern Category: *Structural*

Pattern Name: *Facade*

Design Problem 5. (Calling Remote Object)

"There is a Pizza Company, which has its outlets at various locations. The owner of the company gets a daily report by the staff members of the company from various outlets. The current application supported by the Pizza Company is a desktop application, not a web application. So, the owner has to ask his employees to generate the report and send it to him. But now the owner wants to generate and check the report by his own, so that he can generate it whenever he wants without anyone's help. The owner wants you to develop an application for him. The problem here is that all applications are running at their respective JVMs and the Report Checker application (which we will design soon) should run in the owner's local system. The object required to generate the report does not exist in the owner's system JVM and you cannot directly call on the remote object."

Design pattern Category: *Structural*

Pattern Name: *Proxy*

Design Problem 6. (Problem of country president relationship)

"The problem has to be ensured that a class has only one instance and provides a global point of access to that instance. It is named after the singleton set, which is defined to be a set containing one element. The office of the president of the United States is a Singleton. The United States constitution specifies the means by which a president is elected, limits the term of office, and defines the order of succession. As a result, there can be at most one active presidents at any given time. Regardless of the personal identity of the active president, the title, 'The President of the United States' is a global point of access that identifies the person in the office. "

Design pattern Category: *Creational*

Pattern Name: *Singleton*

Design Problem 7. (Problem of arithmetic expressions)

"To make objects into tree structures and lets clients treat individual objects and compositions uniformly. Although the example is abstract, arithmetic expressions are Composites. An arithmetic expression consists of an operand, an operator (+ - * /), and

another operand. The operand can be a number, or another arithmetic expression. Thus, 2 + 3 and (2 + 3) + (4 * 6) are both valid expressions."

Design pattern Category: *Structural*

Pattern Name: *Composite*

Design Problem 8. (Static binding between an interface and its implementation)

"Hardening of the software arteries has occurred by using sub classing of an abstract base class to provide alternative implementations. This locks in compile-time binding between interface and implementation. The abstraction and implementation cannot be independently extended or composed"

Design pattern Category: *Structural*

Pattern Name: *Proxy*

Design Problem 9. (Problem of lazy initialization)

"The application needs one, and only one, instance of an object. Additionally, lazy initialization and global access are necessary. "

Design pattern Category: *Creational*

Pattern Name: *Singleton*

Design Problem 10. (On demand instantiation of resource hungry objects)

"You need to support resource-hungry objects, and you do not want to instantiate such objects unless and until they are actually requested by the client "

Design pattern Category: *Structural*

Pattern Name: *Proxy*

Design Problem 11. (To manage automobile systems)

"This pattern is found in the sheet metal stamping equipment used in the manufacture of Japanese automobiles. The stamping equipment is an Abstract Factory, which creates auto body parts. The same machinery is used to stamp right hand doors, left hand doors, right front fenders, left front fenders, hoods, etc. for different models of cars

Through the use of rollers to change the stamping dies, the concrete classes produced by the machinery can be changed within three minutes. "

Design pattern Category: *Creational*

Pattern Name: *Abstract Factory*

Design Problem 12. (Architectural modeling for applications)

“A framework needs to standardize the architectural model for a range of applications, but allow for individual applications to define their own domain objects and provide for their instantiation.”

Design pattern Category: *Creational*

Pattern Name: *Factory Method*

Design Problem 13. (Adding new functionality of an object at runtime)

“You want to add behavior or state to individual objects at run-time. Inheritance is not feasible because it is static and applies to an entire class.”

Design pattern Category: *Structural*

Pattern Name: *Decorator*

Design Problem 14. (Development of language irrelevant applications)

“A class of problems occurs repeatedly in a well-defined and well-understood domain. If the domain were characterized with a "language", then problems could be easily solved with an interpretation "engine””.

Design pattern Category: *Behavioral*

Pattern Name: *Interpreter*

Design Problem 15. (Abstraction between data structure and algorithms)

“Need to "abstract" the traversal of wildly different data structures so that algorithms can be defined that are capable of interfacing with each transparently.”

Design pattern Category: *Behavioral*

Pattern Name: *Interpreter*

Design Problem 16. (Abstraction between data structure and algorithms)

“The Company class is the main class that encapsulates several important features related to the system as a whole. We need to ensure that only one instance of this class is present.”

Design pattern Category: *Creational*

Pattern Name: *Factory Method*

Design Problem 17. (Implementation of rollback issue in object’s state)

“Need to restore an object back to its previous state (e.g. "undo" or "rollback" operations). “

Design pattern Category: *Behavioral*

Pattern Name: *Memento*

Design Problem 18. (Controlling lights)

“The Bridge pattern decouples an abstraction from its implementation, so that the two can vary independently. A household switch controlling lights, ceiling fans, etc. is an example of the Bridge. The purpose of the switch is to turn a device on or off. The actual switch can be implemented as a pull chain, simple two position switch, or a variety of dimmer switches.”

Design pattern Category: *Structural*

Pattern Name: *Bridge*

Design Problem 19. (Customer’s ordering systems)

“This problem defines a unified, higher level interface to a subsystem that makes it easier to use. Consumers encounter a Facade when ordering from a catalog. The consumer calls one number and speaks with a customer service representative. The customer service representative acts as a Facade, providing an interface to the order fulfillment department, the billing department, and the shipping department.”

Design pattern Category: *Structural*

Pattern Name: *Façade*

Design Problem 20. (Bidder and auction problem)

“The Observer defines a one-to-many relationship so that when one object changes state, the others are notified and updated automatically. Some auctions demonstrate this pattern. Each bidder possesses a numbered paddle that is used to indicate a bid. The auctioneer starts the

bidding, and "observes" when a paddle is raised to accept the bid. The acceptance of the bid changes the bid price which is broadcast to all of the bidders in the form of a new bid".

Design pattern Category: *Behavioral*

Pattern Name: *Observer*

Design Problem 21. (Designing system Interface)

"The system has an interface named "Media Player". This interface is implemented by a concrete class Audio Player. Audio Player has methods that play mp3 format audio files. There is another interface AdvancedMediaPlayer which is implemented by a concrete class AdvancedAudioPlayer to play vlc amd mp4 format files. It is required to have AudioPlayer class to use AdvancedaudioPlayer class to be able to play other formats."

Design Problem 22. (Vending machine problem)

"The State pattern allows an object to change its behavior when its internal state changes. This pattern can be observed in a vending machine. Vending machines have states based on the inventory, amount of currency deposited, the ability to make a change, the item selected, etc. When currency is deposited and a selection is made, a vending machine will either deliver a product and no change, deliver a product and change, deliver no product due to insufficient currency on deposit, or deliver no product due to inventory depletion."

Design pattern Category: *Behavioral*

Pattern Name: *State*

Design Problem 23. (Air traffic to control several airplanes)

"Design the communications of one plane to the approach of an airport. When a plane is in approach of the airport, it must announce to all the other planes which are around that it intends to be posed, and await their confirmation with all before carrying out the operation. It is the control tower of the airport, which guarantees the regulation of the air traffic, by making sure that there is no trajectory conflict or destination between several planes. Besides the class diagram, represent by a collaboration (diagram of collaboration or diagram of objects and sequence) the landing of a plane between two wanting to land and one wanting to take off."

Design pattern Category: *Behavioral*

Pattern Name: *Mediator*

Design Problem 24. (Children's meals at restaurants)

"The design needs separate the construction of a complex object from its representation so that the same construction process can create different representations. This pattern is used by fast food restaurants to construct children's meals. Children's meals typically consist of a main item, a side item, a drink, and a toy (e.g., a hamburger, fries, Coke, and toy dinosaur). Note that there can be variation in the content of the children's meal, but the construction process is the same. Whether a customer order a hamburger, cheeseburger, or chicken, the process is the same. The employee at the counter directs the crew to assemble a main item, side item, and toys. These items are then placed in a bag. The drink is placed in a cup and remains outside of the bag. This same process is used at competing restaurants."

Design pattern Category: *Creational*

Pattern Name: *Builder*

Design Problem 25. (Bank transactions)

"We get to provides a surrogate or placeholder to provide access to an object. A check or bank draft is a proxy for funds in an account. A check can be used in place of cash for making purchases and ultimately controls access to cash in the issuer's account."

Design pattern Category: *Structural*

Pattern Name: *Proxy*

Design Problem 26. (ATM handling)

"The Chain of Responsibility pattern avoids coupling the sender of a request to the receiver by giving more than one object a chance to handle the request. ATM use the Chain of Responsibility in money giving mechanism."

Design pattern Category: *Behavioral*

Pattern Name: *Chain of Responsibility*

Design Problem 27. (Controlling payment systems for waiters)

“I want to allows requests to be encapsulated as objects, thereby allowing clients to be parameterized with different requests. The "check" at a diner is an example of a Command pattern. The waiter or waitress takes an order or command from a customer and encapsulates that order by writing it on the check. The order is then queued for a short order cook. Note that the pad of ‘checks’ used by each waiter is not dependent on the menu, and therefore they can support commands to cook many different items.”

Design pattern Category: *Behavioral*

Pattern Name: *Command*

Design Problem 28. (To decorate the Christmas tree)

“our application we create a notification bar displaying all the alerts held within a stash. Notification/Alert Collection offers an iterator for the iteration of its components all without showing the Customer how it is instituted the collection. The iterator functionality contains a number of techniques for traversing or altering a collection that can also provide search functions, remove functions etc.”

Design pattern Category: *Behavioral*

Pattern Name: *Iterator*

Design Problem 29. (To construct window GUI interface)

“Design a help manager of a Java application. A help manager allows the show of a help message depending on the objects on which a client has clicked. For example, the "?", sometimes located near the contextual menu in a windows dialog box, allows the show of the help of the button or the area where we click. If the button on which one clicks does not contain help, it is the area containing which displays its help, and so on. If no object contains help, with final, the manager displays "Not help available for this area". Instantiate your class diagram in a sequence diagram of on the example of a printing window. This window (JDialog) consists in an explanatory text (JLabel), and in a container (JPanel). This last contains a Print button (JButton) and a Cancel button (JButton). The Print button contains help ‘Launches the impression of the document’. The Cancel button, the text as well as the window do not contain help. Lastly, the container contains help "Click on one of the buttons". In the sequence diagram, reveal the scenarios: ‘The user asks for the help of the

Print button', 'the user asks for the help of the Cancel button", and 'the user asks for the help of the text."

Design pattern Category: *Behavioral*

Pattern Name: *Chain of Responsibility*

Design Problem 30. (To design a calculator)

"Design a tutorial to learn how to program a calculator. This calculator executes the four basic arithmetic operations. The goal of this tutorial is to make it possible to take a set of operations to be executed sequentially. The tutorial presents a button by an arithmetic operation, and two input fields for the operands. After each click on a button of an operation, the user has then the choice to start again or execute the suite of operations to obtain the result. It is probable that this teach ware evolves in order to make it possible to the user to remove the last operation of the list and to take into account the operation of modulo."

Design pattern Category: *Behavioral*

Pattern Name: *command*

Design Problem 31. (Drawing different Objects)

"A design is composed of graphics (lines, rectangles and roses), positioned at precise positions. Each graphic form must be modeled by a class that provides a method draw (): void. A rose is a complex graphic designed by a "black-box" class component. This component performs this drawing in memory, and provides access through a method getRose (): int that returns the address of the drawing. It is probable that the system evolves in order to draw circles."

Design pattern Category: *Structural*

Pattern Name: *Adapter*

Design Problem 32. (Interface for platforms)

"It is required to use an existing user interface toolkit in developing software applications that work on different platforms. Hence, it is important to include a portable window abstraction in the toolkit such that the user can create a window without being committed to a certain implementation as the window implementation is related to the application platform."

Design pattern Category: *Structural*

Pattern Name: *Bridge*

Design Problem 33. (Handing purchasing requests)

“The system approves purchasing requests. There are four approval authorities and the selection of the approval authority depends on the purchase amount. If the amount of the purchase is higher than one million dollars, the owner is the one who approves. However, if it ranges from 500k to less than one million, the CEO is the one who approves and if it ranges from 25k to less than 500k, the head of the department is the one who approves. Finally, if the purchase is less than 25k, the vice is the one who approves. The system needs to be flexible such that the approval authority for each amount of money can change at run time.”

Design pattern Category: *Behavioral*

Pattern Name: *Chain of Responsibility*

Design Problem 34. (Entries of Menu)

“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. Another menu style prints a letter in front of each string (e.g., A, B, and so on) and lets the user enter a letter to make a choice. Still another menu style prints each string out, and lets the user type in the first few characters of the string to make that 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. It should be extremely easy for us to modify the program so that it uses a different menu style whenever needed.”

Design pattern Category: *Behavioral*

Pattern Name: *Strategy*

Design Problem 35. (Picking Up and Dropping off in Game features)

"The developer of a game desires the player to be able to pick up and drop off a variety of elements which exist in the environment of the game. Two types of these elements are bags and boxes, each of which may contain individual elements as well as other bags and boxes."

Design pattern Category: *Structural*

Pattern Name: *Composite*

Design Problem 36. (Drawing an Image)

“Design a system enabling to draw a graphic image. A graphic image is composed of lines, rectangles, texts and images. An image may be composed of other images, lines, rectangles and texts.”

Design pattern Category: *Structural*

Pattern Name: *Composite*

Design Problem 37. (Display visual objects on a screen)

“Design a system enabling to display visual objects on a screen. A visual object can be composed with one or more texts or images. If needed, the system must allow to add to this object a vertical scrollbar, a horizontal scrollbar, an edge and a menu (these additions may be cumulated).”

Design pattern Category: *Structural*

Pattern Name: *Decorator*

Design Problem 38. (DVD Market)

“Design a DVD market place work. The DVD market place provides DVD to its clients with three categories: children, normal and new. A DVD is new during some weeks, and after change category. The DVD price depends on the category. It is probable that the system evolves in order to take into account the horror category.”

Design pattern Category: *Behavioral*

Pattern Name: *State*

C.1 Sample Code for Preprocessing

```
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer,WordNetLemmatizer
from nltk.tokenize import RegexpTokenizer,word_tokenize
# remove punctuation
def remove_punctuation(data):
    data = data.str.replace('[^\\w\s]','')
    return data
# Converting to lowercase
def text_lowercase(data):
    data = data.apply(lambda x: " ".join(x.lower() for x in x.split()))
    return data
# remove stopwords function
def remove_stopwords(data):
    stop = stopwords.words('english')
    data = data.apply(lambda x: " ".join(x for x in x.split() if x not in stop))
    return data
## Common word removal
def tokenization(data):
    #Instatiate Tokenizer
    tokenizer = RegexpTokenizer(r'\w+')
    data = data.apply(lambda x: tokenizer.tokenize(x))
    return data
# lemmatize string
def lemmatize_word(data):
    lemmatizer = WordNetLemmatizer()
    data = data.apply(lambda x: " ".join([lemmatizer.lemmatize(word) for word in x.split()]))
    return data
```

C.3 Sample Code for Feature Extraction and Evaluation

```
#word Embedding Word2Vec
w2v_data = final_X # Input final raw cleaned data into the pipeline
splitted = []
for row in w2v_data:
    splitted.append([word for word in row.split()])     #splitting words
# Set values for various parameters for the model
num_features = 10    # Word vector dimensionality
min_word_count = 1   # Minimum word count
num_workers = 4      # Number of threads to run in parallel
context = 5          # Context window size
downsampling = 1e-3  # Downsample setting for frequent words
train_w2v = Word2Vec(splitted, workers=num_workers, \
            size=num_features, min_count = min_word_count, \
            window = context, sample = downsampling)
words = list(train_w2v.wv.vocab) # Trained Words in the w2v model
train_w2v.save('test_save.bin') # Save the Model
w2v_data = []    #Load each words with there w2v respecitve Value
vec_w2v = np.zeros(10)
for row in splitted:
    for word in row:
        try:
            vec_w2v = train_w2v.wv[word]
        except:
            pass
    w2v_data.append(vec_w2v)
# summarize the loaded model
vectorizer = TfidfVectorizer(
             use_idf = True,
             smooth_idf = False,
             norm = None,
```

```
            decode_error = 'replace',
            max_features = 5000,
            min_df = 1,
            max_df = 1
            )
tf_idf_train_data = vectorizer.fit_transform(final_X)
print(vectorizer.vocabulary_.__len__())
print(tf_idf_train_data.shape)
##################TF-W2-IDFV
tf_w_data_train = []
tf_idf_train_data_arr = tf_idf_train_data.toarray()
i = 0
print(splitted)
for row in splitted:
    vec = np.zeros(10)
  temp_tfidf = []
    for val in tf_idf_train_data_arr[i]:
      if val != 0:
          temp_tfidf.append(val)
     tf_idf_sum = 0
      for word in row:
       count = 0
       try:
          count += 1
          tf_idf_sum = tf_idf_sum + temp_tfidf[count-1]
          vec = vec + (temp_tfidf[count-1] * train_w2v.wv[word])

       except:
          pass
   vec = (float) (1/tf_idf_sum) * vec
   tf_w_data_train.append(vec)
   i = i + 1
```

C.4 Sample Code for Cosine Similarity (CS)

```
test_data = dp_problem.Y_test #Assigned the test Data
pred_data = dp_class.final_svc_pred #Assgined the predicted Data using SVM learner
wv_test_data = dp_problem.X_test # Assigned vector test Data
wv_train_data = dp_class.tf_w_data_train # Assigned vector orginal Data
 def compare_Test_Target(x,y):
    test_index = []
    target_str = []
    for i in range(len(x)):
        if(x[i]==y[i]):
          test_index.append(i)
          target_str.append(y[i])
    return [test_index,target_str]
def dp_problem_index():
    pre_dp_data = compare_Test_Target(test_data,pred_data)
    print(pre_dp_data)
    dp_problem_indexes = pre_dp_data[0]
    total_indexes_data = []
    for j in range(len(dp_problem_indexes)):
        y = dp_problem_indexes[j]
        v = wv_test_data[y]
        total_indexes_data.append(v)
    return total_indexes_data
def dp_class_names():
    pre_dp_data = compare_Test_Target(test_data,pred_data)
    pred_class_data = pre_dp_data[1]
    total_dp = dp_class.yz_tr.values.tolist()
    total_dp_data = dp_class.tf_w_data_train
    print(pred_class_data)
    dp_creational = []
```

```
dp_creational_data,    dp_structural, dp_structural_data,  dp_behavioral
dp_behavioral_data = []
     for e in range(len(total_dp)):
         if(total_dp[e][0] == "Creational"):
            dp_creational.append(total_dp[e][1])
            dp_creational_data.append(total_dp_data[e])
    for f in range(len(total_dp)):
         if(total_dp[f][0] == "Structural" ):
           dp_structural.append(total_dp[f][1])
           dp_structural_data.append(total_dp_data[f])
    for g in range(len(total_dp)):
         if(total_dp[g][0] == "Behavioral"):
            dp_behavioral.append(total_dp[g][1])
            dp_behavioral_data.append(total_dp_data[g])
    return[dp_creational,
dp_creational_data,dp_structural,dp_structural_data,dp_behavioral,dp_behavioral_data]
def calculate_csm(csm_dp_problem,csm_dp_name):
    dot_product = np.dot(csm_dp_problem, csm_dp_name)
    norm_a = np.linalg.norm(csm_dp_problem)
    norm_b = np.linalg.norm(csm_dp_name)
    csm_final = dot_product /(norm_a*norm_b)
    return csm_final
def csm():
    predicted_test_indexes = compare_Test_Target(test_data,pred_data)
    csm_dp_data_names = dp_class_names()
    csm_test_index = predicted_test_indexes[0]
    csm_dp_problem_data = dp_problem_index()
    csm_creational_data, csm_structural_data, csm_behavioral_data = []
    creat_final,cr = []
    struc_final,st  =[]
    behav_final,bh = []
        for n in range(len(csm_test_index))
        if(test_data[csm_test_index[n]] == "Creational"):
```

```
            for o in range(len(csm_dp_data_names[1])):
csm_creational_data.append(calculate_csm(csm_dp_problem_data[n],csm_dp_data_names
[1][o]))
                creat_temp = []
                creat_temp,csm_creational_data = csm_creational_data,creat_temp
                c = "Dp problem {}".format(csm_test_index[n])
                for elem in range(len(csm_dp_data_names[0])):
                    cr.extend(((csm_dp_data_names[0][elem]),"<===>",creat_temp[elem]))
                creat_final.extend((c,cr))
        elif(test_data[csm_test_index[n]] == "Structural"):
                for o in range(len(csm_dp_data_names[3])):
csm_structural_data.append(calculate_csm(csm_dp_problem_data[n],csm_dp_data_names
[3][o]))
                stru_temp = []
                stru_temp, csm_structural_data =  csm_structural_data,stru_temp
                s = "Dp problem {}".format(csm_test_index[n])
                for elem in range(len(csm_dp_data_names[2])):
                    st.extend(((csm_dp_data_names[2][elem]),"<===>",stru_temp[elem]))
                struc_final.extend((s,st))
        elif(test_data[csm_test_index[n]] == "Behavioral"):
                 for p in range(len(csm_dp_data_names[5])):

csm_behavioral_data.append(calculate_csm(csm_dp_problem_data[n],csm_dp_data_name
s[5][p]))
                behav_temp = []
                behav_temp,csm_behavioral_data = csm_behavioral_data,behav_temp
                b = "Dp problem {}".format(csm_test_index[n])
                for elem in range(len(csm_dp_data_names[4])):
                    bh.extend(((csm_dp_data_names[4][elem]),"<===>",behav_temp[elem]))
                behav_final.extend((b,bh))
    return [creat_final,struc_final,behav_final]
  print(csm())
```

Appendix D: Experiment Results

D.1 Software Design Pattern Class Recommendation Results using classifiers

Table D.1.1 SVM Model Recommended Pattern Class Using Outperform Word2vec Weighted by TFIDF Feature Extraction Method

Design Problem	Correct Pattern Group	Recommended Pattern Group
Design Problem 16	Creational	**Behavioral**
Design Problem 17	Behavioral	**Behavioral**
Design Problem 18	Structural	**Structural**
Design Problem 19	Structural	**Structural**
Design Problem 20	Behavioral	**Behavioral**
Design Problem 21	Structural	**Structural**
Design Problem 22	Behavioral	**Behavioral**
Design Problem 23	Behavioral	**Behavioral**
Design Problem 24	Creational	**Creational**
Design Problem 25	Structural	**Behavioral**
Design Problem 26	Behavioral	**Behavioral**
Design Problem 27	Behavioral	**Behavioral**
Design Problem 28	Behavioral	**Behavioral**
Design Problem 29	Behavioral	**Structural**
Design Problem 30	Behavioral	**Behavioral**
Design Problem 31	Structural	**Structural**
Design Problem 32	Structural	**Structural**
Design Problem 33	Behavioral	**Structural**
Design Problem 34	Behavioral	**Behavioral**
Design Problem 35	Structural	**Structural**
Design Problem 36	Structural	**Structural**
Design Problem 37	Structural	**Structural**
Design Problem 38	Behavioral	**Behavioral**

Table D.1.2 KNN Models Recommended pattern class Using Outperform Word2vec Weighted by TFIDF Feature Extraction Method

Design Problem	Correct Pattern Group	Recommended Pattern Group
Design Problem 16	Creational	**Creational**
Design Problem 17	Behavioral	**Creational**
Design Problem 18	Structural	**Structural**

Design Problem 19	Structural	**Creational**
Design Problem 20	Behavioral	**Behavioral**
Design Problem 21	Structural	**Structural**
Design Problem 22	Behavioral	**Creational**
Design Problem 23	Behavioral	**Creational**
Design Problem 24	Creational	**Creational**
Design Problem 25	Structural	**Structural**
Design Problem 26	Behavioral	**Behavioral**
Design Problem 27	Behavioral	**Behavioral**
Design Problem 28	Behavioral	**Behavioral**
Design Problem 29	Behavioral	**Creational**
Design Problem 30	Behavioral	**Behavioral**
Design Problem 31	Structural	**Structural**
Design Problem 32	Structural	**Structural**
Design Problem 33	Behavioral	**Behavioral**
Design Problem 34	Behavioral	**Structural**
Design Problem 35	Structural	**Structural**
Design Problem 36	Structural	**Behavioral**
Design Problem 37	Structural	**Creational**
Design problem 38	Behavioral	**Creational**

Table D.1.3 NB Model recommended pattern class using Outperform Word2vec Weighted by TFIDF Feature Extraction Method

Design Problem	**Correct Pattern Group**	**Recommended Pattern Group**
Design Problem 16	Creational	**Creational**
Design Problem 17	Behavioral	**Behavioral**
Design Problem 18	Structural	**Structural**
Design Problem 19	Structural	**Structural**
Design Problem 20	Behavioral	**Behavioral**
Design Problem 21	Structural	**Structural**
Design Problem 22	Behavioral	**Behavioral**
Design Problem 23	Behavioral	**Behavioral**
Design Problem 24	Creational	**Behavioral**
Design Problem 25	Structural	**Structural**
Design Problem 26	Behavioral	**Structural**
Design Problem 27	Behavioral	**Structural**
Design Problem 28	Behavioral	**Structural**
Design Problem 29	Behavioral	**Behavioral**
Design Problem 30	Behavioral	**Behavioral**

Design Problem 31	Structural	**Behavioral**
Design Problem 32	Structural	**Structural**
Design Problem 33	Behavioral	**Behavioral**
Design Problem 34	Behavioral	**Behavioral**
Design Problem 35	Structural	**Structural**
Design Problem 36	Structural	**Behavioral**
Design Problem 37	Structural	**Behavioral**
Design Problem 38	Behavioral	**Behavioral**

Table D.1.4 RF Models recommended pattern class using outperform Feature Extraction model (TF-IDF or word2vec)

Design Problem	**Correct Pattern Group**	**Recommended Pattern Group**
Design Problem 16	Creational	**Creational**
Design Problem 17	Behavioral	**Creational**
Design Problem 18	Structural	**Structural**
Design Problem 19	Structural	**Creational**
Design Problem 20	Behavioral	**Creational**
Design Problem 21	Structural	**Structural**
Design Problem 22	Behavioral	**Creational**
Design Problem 23	Behavioral	**Creational**
Design Problem 24	Creational	**Creational**
Design Problem 25	Structural	**Structural**
Design Problem 26	Behavioral	**Behavioral**
Design Problem 27	Behavioral	**Creational**
Design Problem 28	Behavioral	**Behavioral**
Design Problem 29	Behavioral	**Creational**
Design Problem 30	Behavioral	**Behavioral**
Design Problem 31	Structural	**Structural**
Design Problem 32	Structural	**Structural**
Design Problem 33	Behavioral	**Behavioral**
Design Problem 34	Behavioral	**Structural**
Design Problem 35	Structural	**Structural**
Design Problem 36	Structural	**Behavioral**
Design Problem 37	Structural	**Creational**
Design Problem 38	Behavioral	**Creational**

D.2 Cosine Similarity (CS) measures values for 35 design problems related to GoF Design Pattern Collection

Table D.2 Cosine Similarity Results for Design Problems recommended via outperformed SVM

Design Problem ID	Recommended DP Group	Correct Pattern	1^{st} Selected Pattern	CS_1	2^{nd} Selected pattern	CS_2	3^{rd} Selected pattern	CS_3
13	**Structural**	Decorator	**Decorator**	**0.344**	Flyweight	0.123	Composite	0.093
15	**Behavioral**	Interpreter	***Visitor***	**0.125**	Memento	0.039	Interpreter	0.002
17	**Behavioral**	Memento	***Chain of Responsibility***	**0.085**	State	0.071	Command	0.032
18	**Structural**	Bridge	**Bridge**	**0.132**	Adapter	0.092	Flyweight	0.032
19	**Structural**	Facade	**Facade**	**0.641**	Bridge	0.249	Composite	0.241
20	**Behavioral**	Observer	**Observer**	**0.411**	Chain of Responsibility	0.331	state	0.127
21	**Structural**	Adapter	**Adapter**	**0.631**	Bridge	0.083	Proxy	0.019
22	**Behavioral**	State	**State**	**0.231**	Command	0.182	Iterator	0.051
23	**Behavioral**	command	**command**	**0.763**	Mediator	0.191	State	0.021
24	**Creational**	Builder	**Builder**	**0.612**	Prototype	0.043	Abstract Factory	0.008
26	**Behavioral**	Chain of Responsibility	***Template Method***	**0.048**	Command	0.009	Memento	0.002
27	**Behavioral**	command	**command**	**0.321**	Chain of Responsibility	0.173	Mediator	0.098
28	**Behavioral**	Iterator	**Iterator**	**0.451**	Visitor	0.365	Interpreter	0.056

30	**Behavioral**	Command	**Command**	**0.342**	State	0.222	Visitor	0.004
31	**Structural**	Adapter	**Adapter**	**0.291**	Facade	0.213	Proxy	0.089
32	**Structural**	Bridge	**Bridge**	**0.156**	Flyweight	0.089	Facade	0.043
34	**Behavioral**	Strategy	**Strategy**	**0.651**	State	0.316	Command	0.024
35	**Structural**	Composite	**Composite**	**0.189**	Facade	0.097	Decorator	0.011
36	**Structural**	Composite	**Composite**	**0.365**	Decorator	0.123	Bridge	0.008
37	**Structural**	Decorator	**Decorator**	**0.512**	Composite	0.399	Facade	0.066
38	**Behavioral**	State	**State**	**0.492**	Template Method	0.236	Strategy	0.007