Design Pattern Recommendation Using Doc2vec

Nada Shorim[1,2], Mohammad El-Ramly[2], Hanaa Mobarz[2]
[1]Misr International University, Cairo, Egypt
{nada.ayman}@miuegypt.edu.eg
[2]Faculty of Computers and Artificial Intelligence, Cairo University, Egypt
{m.elramly, h.mobarz}@fci-cu.edu.eg

***Abstract*—Design patterns are widely recognized as valuable tools in software engineering for improving software quality and reducing development time. Selecting a suitable design pattern is crucial for creating high-quality software systems. The wrong design pattern can lead to difficult-to-maintain code. The appropriate design pattern can improve the overall quality of the software, making it easier to maintain, modify, and extend over time. However, selecting the appropriate design pattern for a specific design problem from textual descriptions is challenging and requires a deep understanding of their functionality and characteristics. In order to address this challenge, a recommendation system is developed that utilizes text classification through the application of doc2vec. This approach has not been previously explored in research and has the potential to be highly effective. This approach involves: first, preprocessing techniques such as stop word removal, tokenization, and stemming are applied to design pattern category descriptions, design patterns' descriptions, and problem scenarios. Then doc2vec is applied for word embeddings to create the model. The study evaluates two approaches for selecting design patterns. The first approach focuses solely on making a recommendation based on the descriptions of the design patterns themselves and achieves an accuracy of 54.3%, making accurate recommendations for 25 out of 46 problems. The second approach considers the category of the design patterns before making a selection, resulting in an enhanced accuracy of 65.21% and correctly recommending a design pattern for 30 out of 46 problems. The second approach outperforms the first by leveraging the design pattern category, significantly narrowing the search space and improving recommendation accuracy. These initial results are promising and pave the way for further improvement by enhancing the technique or combining doc2vec with other techniques.**

***Index Terms*—Design Patterns, Design Patterns Recommendation, NLP, doc2vec, Preprocessing, Text Similarity**

I. Introduction

Design patterns gained wide popularity since the "Gang of Four" (GOF) released their book "Design Patterns: Elements of Reusable Object-Oriented Software" [1] in 1994.

A design pattern is a general, repeatable solution to a frequently occurring issue in software design. It helps in designing software systems to support change rather than fight it. Design patterns make requirement changes more manageable and the code easier to read and understand. Design patterns also provide faster development and fewer bugs.

979-8-3503-2210-1/23/$31.00 ©2023 IEEE

According to Wedyan and Abufakher [2], applying design patterns has many benefits, with software quality improvement being the main benefit. Also, Design patterns improve the maintainability of the software.

According to Thabasum et al. [3], 65% of the decisions made during the software design phase have an immediate impact on the work done during the phases of the software development cycle. Therefore, selecting a suitable design is essential. However, choosing a suitable design pattern can be challenging for several reasons. Firstly, numerous design patterns are available, and selecting the most appropriate one for a specific design problem requires a deep understanding of each pattern's functionality [4]. Secondly, a design pattern that works well in one context may not be suitable for another. Thus, the selection process must consider the specific environment in which the pattern will be applied. Additionally, the pattern description's heterogeneity adds to the selection process's complexity [5]. Overall, choosing a suitable design pattern requires a comprehensive understanding of the available patterns, their functionality, and the specific context in which they will be applied.

This paper presents a novel approach for selecting the most appropriate design pattern for a given problem scenario in software engineering. The proposed approach involves developing a recommendation system that uses Natural Language Processing (NLP) and machine learning techniques to analyze the text description of the problem and recommend the most suitable design pattern. The system uses text preprocessing techniques such as stop word removal, tokenization, stemming, and word embedding to represent the text numerically.

The description is entered as text to provide a problem scenario as input to a recommendation system. Word embedding, also known as text representation, is a method utilized in NLP to represent the meaning of words and facilitate the processing of text by machines. This method converts text into numerical values, enabling the machine to more efficiently comprehend and analyze the text.

One of the most popular word embedding techniques is word2vec [6]. Word2vec transforms individual words into a numerical representation, i.e. a vector. Word2vec takes a corpus as an input, creating a vector associated with each word in the corpus. The vector created then goes through a neural network. The neural network is trained using a single hidden layer to calculate the embedding weight.

Doc2vec [7] is an extended version of word2vec [6]. Doc2vec is used to create a representation for documents, not just words like word2vec. Doc2vec has an additional vector which is the document vector. Doc2Vec can be structured using two different approaches: (1) Distributed Bag of Words: Neural network is trained to predict the probability distribution of words by creating a paragraph vector given a randomly chosen word from the paragraph. (2) Distributed Memory: By training a neural network to infer a center word from context words and a context paragraph, the paragraph vectors are created.

Given that our objective involves recommending design patterns based on input text descriptions of problem scenarios, it is more appropriate to use doc2vec rather than word2vec. This is because doc2vec is designed to capture the semantic meaning of a document as a whole, rather than just individual words. Therefore, doc2vec is better suited for our objective since it can effectively represent the relationships between design patterns, their descriptions, and the input problem scenarios.

In this paper, we describe our approach and experiments for design pattern recommendation using doc2vec, based on textual descriptions of the problem scenario, design pattern category and design pattern description. Our results and findings suggest that this approach is promising and can aid developers and designers in selecting the suitable pattern, with room for further enhancements or combination with other existing methods.

The paper is organized as follows: Section II describes the related work. The dataset used is described in Section III. The proposed approach is defined in Section IV. Experiments on the approach and results are presented in Section V. The conclusions of the research are presented in Section VI.

II. Related Work

A lot of research has been done on the selection of suitable design patterns for certain design problem scenarios. According to a survey by Naghdipour et al. [8] [9], the text classification approach is the best approach for design pattern selection, along with the ontology approach.

A. Text Classification

Hasheminejad and Jalili [10] proposed a text classification approach which starts with preprocessing (removing stop word, stemming) the problem definition, then preparing a classifier model and finally, the correct design pattern is suggested. Different classifiers were tried, like SVM, Naïve Bayes, and KNN. SVM achieved the best result with a precision of 62% when evaluated on 19 design problems.

Hamdy and Elsayed [11] proposed a text classification method for an automatic selection of software design patterns. Design patterns description collected from GOF [1] and design problem are first preprocessed by applying tokenization, stop word removal and stemming. Vector Space Model (VSM) is created, and each design pattern is represented as a vector of features (Unigram and Bigram). Then the performance of the text retrieval process is improved using TF*IDF. Cosine Similarity is used to find a suitable design pattern. This approach is evaluated by trying 32 real design problems and achieved a precision of 65.5%.

Hamdy and Elsayed [12] suggested a text classification method for an automated software design pattern selection system. Design patterns description collected from GOF [1] and design problem are first preprocessed by applying tokenization, stop word removal and stemming. A Vector Space Model (VSM) of unigrams is generated after preprocessing. A single vector is used to represent every single pattern. Then the performance of the text retrieval process is improved using TF*IDF. Along with the VSM, Topic modelling (Latent Dirichlet Allocation) is also used, the LDA model is trained, and topics are extracted from the design patterns description. The LDA VSM and the Unigram VSM are combined. Cosine Similarity is used to find a suitable design pattern. This approach is evaluated by trying 29 real design problems and achieved a precision of 72%.

Rahmati et al. [13] used an extended version of VSM and explicit semantic analysis (ESA) to determine the similarities between the design challenge and the design patterns. The semantic similarity between the user-entered text and a collection of keywords taken from each text in the repository of patterns is determined by ESA. In the extended version of VSM, using the WordNet dictionary, each word is taken into account together with a set of synonyms and hyponyms to determine its weight. TF*IDF is applied to ESA and the extended VSM. Cosine similarity is used for selecting a suitable design pattern. This approach is evaluated by trying 38 design problems and achieved a precision of 73.52%.

Hamdy and Elsayed [14] suggested a text classification method for an automated software design pattern selection system. Design patterns description from GOF [1] and design problem are first preprocessed by applying tokenization, stop word removal and stemming. After preprocessing, a Vector Space Model (VSM) of unigrams is created. Each design pattern is represented as a vector. Then the efficiency of the text retrieval process is enhanced by TF*IDF. Along with the VSM, topic modelling using Latent Dirichlet Allocation LDA) is also used. Topics are extracted from the description of design patterns using a trained LDA model. LDA, VSM and the Unigram VSM are combined. Cosine Similarity is used to find a suitable design pattern. This approach is evaluated by trying 29 real design problems and achieved a precision of 72%.

B. Ontology

Bou et al. [15] proposed an automatic method for recommending and ranking design patterns to ease the selection of design pattern and continued their work in [16]. An input design problem and the design patterns are scored for similarity. This approach constructs two vectors: Design Pattern Vector (DPV) and Input Problem Vector (IPV). DPV is

represented by the types of problems solved by design patterns according to Kampffmeyer's Design Pattern Intent Ontology (DPIO) [17]. The IPV is an input design problem description. For each pattern, the cosine similarity between the IPV and the DPV is calculated. Following that, design patterns are ranked using the calculated similarity scores. The approach in [15] is evaluated by experimenting with 24 input problem descriptions. The approach achieved an accuracy of 41.67% and 91.67% with top-one rank and top-five rank. The approach in [16] is evaluated by experimenting with 36 input problem descriptions. The approach achieved an accuracy of 47.22% and 88.89% with top-one and top-five ranks.

In conclusion, it is evident from the related work that most research studies in design pattern selection have evaluated their proposed approaches using a limited number of cases. Moreover, it is noteworthy that none of the previous studies have evaluated the effectiveness of doc2vec in design pattern selection. This highlights the need for further research to evaluate the proposed approach using a larger dataset and to investigate the effectiveness of other text representation techniques in design pattern selection.

III. Dataset

GOF described 23 pattern categorized into three groups: behavioral, structural, and creational, as shown in Fig. 1, classified based on purpose. Design patterns are not considered finished designs to be used directly in source code but as ready-made templates. According to GOF [1], each of these templates is documented in a format that holds mainly a 'pattern name', 'intent', 'motivation', 'structure', 'participants', 'known uses', 'implementation' and 'a sample code'.

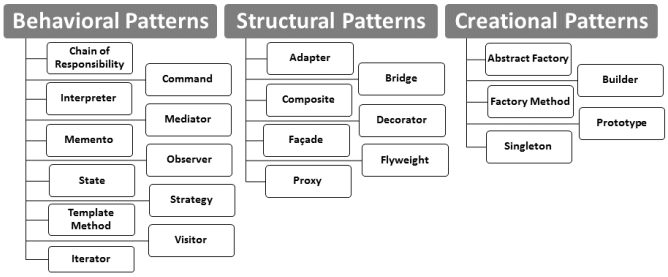

Fig. 1. GOF Design Patterns Categories

Three corpus were collected to evaluate the approach, a corpus for design pattern categories description, a corpus for design patterns description, and a corpus for problem scenarios. Both category and design pattern descriptions were collected from GOF [1]. The problem scenarios is collected from "Head First Design Patterns: A Brain-Friendly Guide" [18] and from Sarcar and Metsker [19]. Forty-six problem scenarios are collected, two for each design pattern. According to our knowledge, this is largest corpus of problem scenarios used in this research, so far. Samples from the problem scenarios are listed below:

Sample Problem 1: "In an application, you may have different database users. For example, one user may use Oracle, and the other may use SQL Server. Whenever you need to insert data into your database, you need to create either a SqlConnection or an OracleConnection and only then can you proceed. If you put the code into if-else (or switch) statements, you need to repeat a lot of code, which isn't easily maintainable. This is because whenever you need to support a new type of connection, you need to reopen your code and make those modifications. This type of problem can be resolved using the Factory Method pattern. Here I'll provide an abstract creator class (IAnimalFactory) to define the basic structure. As per the definition, the instantiation process will be carried out through the subclasses that derive from this abstract class." Suggested pattern: ***Factory Method Design Pattern - Creational Category***

Sample Problem 2: "In the world of computer science, consider a simple UI-based example. This UI is connected to some database. A user can execute some query through that UI, and after searching the database, the result is returned in the UI. With this pattern, you segregate the UI from the database. If a change occurs in the database, the UI should be notified so that it can update its display according to the change. To simplify this scenario, assume that you are the person responsible for maintaining a particular database in your organization. Whenever there is a change made to the database, you want to get a notification so that you can take action if necessary." Suggested pattern: ***Observer Design Pattern - Behavioral Category***

Sample Problem 3: "When need to add to some shape component a border, or a shadow functionality." Suggested pattern: ***Decorator Design Pattern - Structural Category***

Sample Problem 4: "In a game where we can have different characters and each character can have multiple weapons to attack but at a time can use only one weapon. The method attack() will have different implementation depends on which weapon is being used." Suggested pattern: ***Strategy Design Pattern - Behavioral Category***

Sample Problem 5: "When drawing a lot of shapes with different colors: one object for red circle, one object for blue circle and so on. In case red circle was already created once, there is no need to create new such object, since the same object may be reused." Suggested pattern: ***Flyweight Design Pattern - Structural Category***

IV. Approach

This paper evaluates two different approaches for recommending design patterns. The first approach is a simple one that operates at the level of design patterns. As illustrated in Fig. 2, both the text of the problem and the description of the design patterns undergo preprocessing. A model is then created using doc2vec and trained on the preprocessed design pattern descriptions. The problem description is then passed through the trained model, and the recommended design pattern is computed using cosine similarity. Cosine similarity is a widely

Fig. 2. Design pattern recommendation approach - Design pattern level

used metric for measuring similarity and it is used in the related work [11, 12, 13, 14]

The second approach is similar to the first, but with the addition of design pattern categories, as depicted in Fig. 3. It involves three texts that undergo preprocessing: the problem description, the design pattern categories description, and the design pattern description.

- At the category level, a doc2vec model is created and trained on the preprocessed design pattern category descriptions. The problem description is passed through the trained model, and cosine similarity is used to select the most 'N' nearest categories.
- At the design pattern level, only the design patterns falling under the selected category(s) from the previous level are used to train the model. This results in less processing. The problem description is then passed through the trained model, and cosine similarity is used to recommend the most suitable design pattern.

A. Preprocessing

Text preprocessing is applied to the three corpus described in section III. Preprocessing the text before using it in the model is very important. It is used to feed the model only the texts that matter.

1) Noise Removal: is applied to remove all noises from text, like punctuation such as [. , :] and stop words such as [a, an, the, etc.]. Noise removal is applied using the NLTK library [20].

2) Tokenization: It involves breaking the text down into a set of meaningful pieces. These pieces are known as tokens. Tokenization is used for splitting down paragraphs and sentences into smaller units that can be more easily assigned meaning. Tokenization is applied using the NLTK library [20].

3) Stemming: is used to standardize the tokens to their base or root forms. As an example, the words "learn", "learning", and "learned" will all be stemmed from the same root word, "learn". Stemming is applied using PorterStemmer from NLTK library [20].

A complete example of preprocessing applied on Sample Problem number 3:

- Original Text: "When need to add to some shape component a border, or a shadow functionality."
- Punctuation removal: "When need to add to some shape component a border or a shadow functionality"
- Tokenization: ["When", "need", "to", "add", "to", "some", "shape", "component", "a", "border", "or", "a", "shadow", "functionality"]
- Stop word removal: ["need", "add", "shape", "component", "border", "shadow", "functionality"]
- Stemming: ["need", "add", "shape", "compon", "border", "shadow", "function"]

A complete example of preprocessing applied on Sample Problem number 4:

- Original Text: "In a game where we can have different characters and each character can have multiple weapons to attack but at a time can use only one weapon. The method attack() will have different implementation depends on which weapon is being used"
- Punctuation removal: "In a game where we can have different characters and each character can have multiple weapons to attack but at a time can use only one weapon The method attack will have different implementation depends on which weapon is being used"
- Tokenization: ["In", "a", "game", "where", "we", "can", "have", "different", "characters", "and", "each", "character", "can", "have", "multiple", "weapons", "to", "attack", "but", "at", "a", "time", "can", "use", "only", "one", "weapon", "The", "method", "attack", "will", "have", "different", "implementation", "depends", "on", "which", "weapon", "is", "being", "used"]
- Stop word removal: ["game", "different", "characters", "character", "multiple", "weapons", "attack", "time", "use", "one", "weapon", "method", "attack", "different", "implementation", "depends", "weapon", "used"]
- Stemming: ["game", "differ", "charact", "charact", "multipl", "weapon", "attack", "time", "use", "one", "weapon", "method", "attack", "differ", "implement", "depend", "weapon", "use"]

A complete example of preprocessing applied on Sample Problem number 5:

- Original Text: "When drawing a lot of shapes with different colors: one object for red circle, one object for blue circle and so on. In case red circle was already created once, there is no need to create new such object, since the same object may be reused."
- Punctuation removal: "When drawing a lot of shapes with different colors one object for red circle one object for blue circle and so on In case red circle was already created once there is no need to create new such object

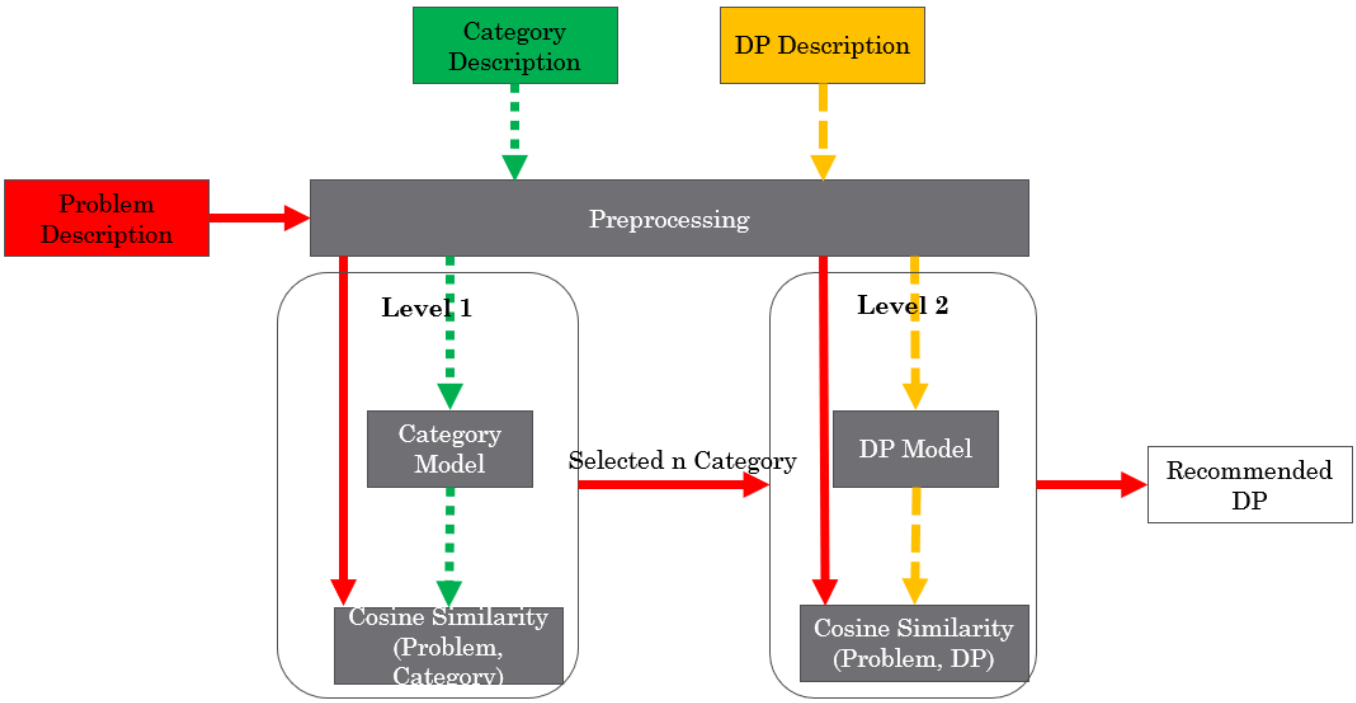

Fig. 3. Design pattern recommendation approach - Category level to design pattern level

since the same object may be reused"

- Tokenization: ["When", "drawing", "a", "lot", "of", "shapes", "with", "different", "colors", "one", "object", "for", "red", "circle", "one", "object", "for", "blue", "circle", "and", "so", "on", "In", "case", "red", "circle", "was", "already", "created", "once", "there", "is", "no", "need", "to", "create", "new", "such", "object", "since", "the", "same", "object", "may", "be", "reused"]
- Stop word removal: ["drawing", "lot", "shapes", "different", "colors", "one", "object", "red", "circle", "one", "object", "blue", "circle", "case", "red", "circle", "already", "created", "need", "create", "new", "object", "since", "object", "may", "reused"]
- Stemming: ["draw", "lot", "shape", "differ", "color", "one", "object", "red", "circl", "one", "object", "blue", "circl", "case", "red", "circl", "alreadi", "creat", "need", "creat", "new", "object", "sinc", "object", "may", "reus"]

B. Doc2vec Model

Doc2vec [7] is used when dealing with an extensive text corpus. It embeds the words by creating a vector space model with several dimensions. Doc2vec uses a distributed bag of words but with an addition to paragraph vector (PV-DBOW), as shown in Fig. 4.

The paragraph ID was added to stand in for any context-specific information that was absent from a document. For words drawn at random from the word embeddings in the paragraph, a paragraph ID is utilized.

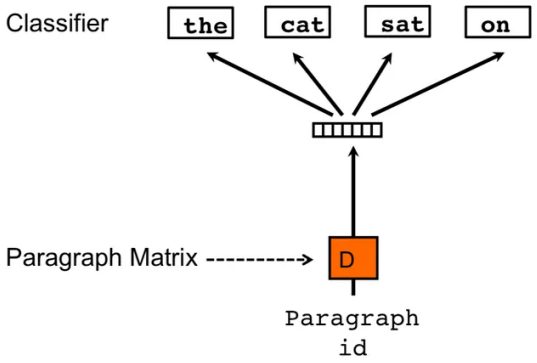

Fig. 4. Distributed Bag of Words Version of Paragraph Vector (PV-DBOW)

Using a target word as a starting point, the Distributed Bag-Of-Words (DBOW) model helps infer the context words. The target word is used in the distributed bag of words model to roughly represent the word's context.

V. Experiments & Results

The following parameters were configured while conducting the experiments:

- dm = 0; 'Distributed memory' (PV-DM) is used if dm = 1. A distributed bag of words (PV-DBOW) is used otherwise.
- min_count = 2; Ignores all words with total frequency lower than this.

- epochs = 40; Number of iterations (epochs) over the corpus.

The first experiment was conducted using the design pattern level approach. This approach correctly recommended 25 problems out of 46, resulting in an accuracy rate of 54.3%.

The second approach comprises two experiments. In the first experiment, the top 'one' category is selected at the category level for each problem. In the second experiment, the top 'two' categories are selected at the category level for each problem. The results of these experiments are presented in Table I.

TABLE I
TWO LEVEL APPROACH RESULTS

	Level 1 Category	**Level 2 DP**
Top 1 Category	25 54.30%	14 30.40%
Top 2 Category	44 95.60%	**30** **65.21%**

In the first experiment where 'one' category was selected, 25 problems were recommended with the correct category, with results of 54.3%. The model at the design pattern level is trained only on the design pattern description in that category. Fourteen problems were recommended correctly, with results of 30.4%. The sample problems' results are shown in table II.

TABLE II
CATEGORY LEVEL TO DESIGN PATTERN LEVEL APPROACH - TOP 1 CATEGORY

Problem	**Suitable Category & Design Pattern**	**Level 1 Results (Top 1 Categories)**	**Level 2 Results (Top 1 DP)**
1	Creational Factory Method	Structural	Bridge
2	Behaviour Observer	Behaviour	Observer
3	Structural Decorator	Structural	Decorator
4	Behavioral Strategy	Cretional	Abstract Factory
5	Structural Flyweight	Creational	Factory Method

In the second experiment, where two categories were selected, 44 problems were recommended with the correct category, with results of 95.6%. The model at the design pattern level is trained only on the design patterns' description in that two categories. Thirty problems were recommended correctly, with results of 65.21%. The sample problems' results are shown in table III.

By comparing the results in Tables II and III, it can be seen that problem 1 in Table II is classified in the wrong category. Consequently, at the design pattern level, it was recommended to wrong design pattern. While in Table III, when the top 'two' categories were selected, it was classified in the correct category. This led to the recommendation of the correct design pattern.

Table IV presents the results obtained by this paper and the related work. It is important to note that simply comparing the accuracy/score of each paper is insufficient since they used varying numbers of cases in their evaluations and different datasets. Therefore, it is crucial to consider the number of correctly recommended cases.

TABLE III
CATEGORY LEVEL TO DESIGN PATTERN LEVEL APPROACH - TOP 2 CATEGORY

Problem	**Suitable Category & Design Pattern**	**Level 1 Results (Top 2 Categories)**	**Level 2 Results (Top 1 DP)**
1	Creational Factory Method	Structural Creational	Factory Method
2	Behaviour Observer	Behaviour Creational	Observer
3	Structural Decorator	Structural Creational	Decorator
4	Behavioral Strategy	Cretional Behaviour	Abstract Factory
5	Structural Flyweight	Creational Structural	Adapter

TABLE IV
COMPARISON WITH RELATED WORK

Ref	**Cases**	**Correctly Recommended**	**%**
Hasheminejad and Jalili [10]	19	12	62%
Hamdy and Elsayed [11]	32	20	65.5%
Hamdy and Elsayed [12]	29	21	72%
Rahmati et al. [13]	38	28	73.52%
Hamdy and Elsayed [14]	29	21	72%
Bou et al. [15]	24	10	41.67%
Laosen et al. [16]	36	17	47.22%
Doc2Vec Approach	46	**30**	65.21%

As shown in Table IV, our approach achieved the highest number of correctly recommended cases, with 30 problems accurately matched to their suitable design pattern. This improvement can be attributed to our strategy of working at the category level initially. In the first experiment, only 25 cases were correctly recommended, indicating the effectiveness of this approach.

However, it is worth noting that some previous studies have achieved higher accuracies than our approach. This indicates that there is still room for improvement in our methodology.

VI. CONCLUSION

Design patterns highly affect software quality and can help in maintainability and save time. Many software engineers, especially at junior level, do not have enough experience to choose a suitable design pattern.

This paper proposes an approach to facilitate the selection of the most appropriate design pattern for a given problem scenario. The proposed approach involves developing a recommendation system that takes a text description of the problem as input. The text is then preprocessed using techniques such as stop word removal, tokenization, and stemming, to ensure optimal text representation.

Furthermore, word embedding techniques, such as doc2vec, are applied to represent the text in a numerical form. The

resulting model is then used to recommend the most suitable design pattern for the given problem scenario. The proposed recommendation system aims to address the challenges associated with manually selecting a design pattern by leveraging the power of natural language processing and machine learning techniques.

Two approaches were evaluated in this paper. Both approaches start with preprocessing the text by removing punctuation and stop words, then tokenising the text and stem it to its root. Doc2vec is then used to create representations of the documents(design patterns description) using a neural network and Distributed Bag of Words. Finally, cosine similarity is used to compute the suitable design pattern.

The first approach is evaluated, which works on the design pattern level only where a problem scenario text goes through preprocessing as well as design pattern description corpus from GOF [1] and then doc2vec model is created using the corpus, the problem scenario text is compared with the doc2vec model. Finally, the most suitable design pattern is recommended. This approach correctly recommended 25 cases.

The second approach is evaluated, which works on the category level and then goes to the design pattern level, where a problem scenario text goes through preprocessing as well as design pattern description corpus and category corpus from GOF [1]. First, the doc2vec model is created using the category corpus; the problem scenario text is compared with the doc2vec model, and the suitable category is recommended. According to the category recommended, another doc2vec model is created with the design pattern description in that category. Finally, the most suitable design pattern is recommended. This approach correctly recommended 30 cases.

By considering the design pattern category first, the recommendation system's accuracy was improved from 25 to 30 correctly recommended cases. This improvement suggests that considering the design pattern category can significantly narrow down the search space and increase the accuracy of the recommendation. This finding highlights the importance of carefully considering the design pattern category when selecting the most appropriate design pattern for a given problem scenario.

As part of future work, it is possible to enhance the results by combining doc2vec with other techniques. For instance, it can be integrated with other text representation techniques or machine learning algorithms to improve the accuracy of the recommendations. Also for the future work, more problems scenario is to be collecting for better evaluation.

REFERENCES

[1] Johnson Gamma Helm and Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. 1994.

[2] Fadi Wedyan and Somia Abufakher. "Impact of design patterns on software quality: a systematic literature review". In: *IET Software* 14.1 (2020), pp. 1–17. DOI: https://doi.org/10.1049/iet-sen.2018.5446.

[3] S Saira Thabasum and Mani Sundar. "A survey on software design pattern tools for pattern selection and implementation". In: *International Journal of Computer Science & Communication Networks (IJCSCN)* 2.4 (2012), pp. 496–500.

[4] Grady Booch. "On architecture". In: *IEEE software* 23.2 (2006), pp. 16–18.

[5] Perla Velasco-Elizondo et al. "Knowledge representation and information extraction for analysing architectural patterns". In: *Science of Computer Programming* 121 (2016), pp. 176–189.

[6] Tomas Mikolov et al. "Efficient estimation of word representations in vector space". In: *arXiv preprint arXiv:1301.3781* (2013).

[7] Quoc Le and Tomas Mikolov. "Distributed representations of sentences and documents". In: *International conference on machine learning*. PMLR. 2014, pp. 1188–1196.

[8] Amene Naghdipour, Seyed Mohammad Hossien Hasheminejad, and Mohammad Reza Keyvanpour. "DPSA: A Brief Review for Design Pattern Selection Approaches". In: *2021 26th International Computer Conference, Computer Society of Iran (CSICC)*. IEEE. 2021, pp. 1–6.

[9] Ameneh Naghdipour, Seyed Mohammad Hossein Hasheminejad, and Roghayeh Leila Barmaki. "Software design pattern selection approaches: A systematic literature review". In: *Software: Practice and Experience* 53.4 (2023), pp. 1091–1122.

[10] Seyed Mohammad Hossein Hasheminejad and Saeed Jalili. "Design patterns selection: An automatic two-phase method". In: *Journal of Systems and Software* 85.2 (2012), pp. 408–424.

[11] Abeer Hamdy and Mohamed Elsayed. "Automatic Recommendation of Software Design Patterns: Text Retrieval Approach." In: *J. Softw.* 13.4 (2018), pp. 260–268.

[12] Abeer Hamdy and Mohamed Elsayed. "Topic modelling for automatic selection of software design patterns". In: *Proceedings of the International Conference on Geoinformatics and Data Analysis*. 2018, pp. 41–46.

[13] Raheleh Rahmati, Abbas Rasoolzadegan, and Diyana Tehrany Dehkordy. "An Automated Method for Selecting GoF Design Patterns". In: *2019 9th International Conference on Computer and Knowledge Engineering (ICCKE)*. IEEE. 2019, pp. 345–350.

[14] Abeer Hamdy and Mohamed ElSayed. "Towards More Accurate Automatic Recommendation of Software Design Patterns." In: *Journal of Theoretical & Applied Information Technology* 96.15 (2018).

[15] Channa Bou, Nasith Laosen, and Ekawit Nantajeewarawat. "Design Pattern Ranking Based on the Design Pattern Intent Ontology". In: *Asian Conference on Intelligent Information and Database Systems*. Springer. 2018, pp. 25–35.

[16] Nasith Laosen, Channa Bou, and Ekawit Nantajeewarawat. “Automatic recommendation of design patterns based on patterns’ intent”. In: *International Journal of Innovative Computing, Information and Control* 16.4 (2020), pp. 1147–1163.

[17] Holger Kampffmeyer and Steffen Zschaler. “Finding the pattern you need: The design pattern intent ontology”. In: *International Conference on Model Driven Engineering Languages and Systems*. Springer. 2007, pp. 211–225.

[18] Eric Freeman et al. *Head First Design Patterns: A Brain-Friendly Guide*. ” O’Reilly Media, Inc.”, 2004.

[19] Vaskaran Sarcar and Steven John Metsker. *Design Patterns in C#*. Springer, 2004.

[20] Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python: analyzing text with the natural language toolkit*. ” O’Reilly Media, Inc.”, 2009.