Model-Driven Engineering and Machine Learning for Legacy System Modernization

Hamza Abdelmalek
GLISI Team, Faculty of Sciences and Techniques of Errachidia, Moulay Ismail University, Morocco
h.abdelmalek@edu.umi.ac.ma (corresponding author)

Zakaria Babaalla
GLISI Team, Faculty of Sciences and Techniques of Errachidia, Moulay Ismail University, Morocco
za.babaalla@edu.umi.ac.ma

Charaf Ouaddi
GLISI Team, Faculty of Sciences and Techniques of Errachidia, Moulay Ismail University, Morocco
c.ouaddi@edu.umi.ac.ma

Lamya Benaddi
GLISI Team, Faculty of Sciences and Techniques of Errachidia, Moulay Ismail University, Morocco
l.benaddi@edu.umi.ac.ma

Abdeslam Jakimi
GLISI Team, Faculty of Sciences and Techniques of Errachidia, Moulay Ismail University, Morocco
ajakimi@yahoo.fr

Received: 22 November 2025 | Revised: 13 December 2025 | Accepted: 26 December 2025

 DOI: https://doi.org/10.48084/etasr.16444

ABSTRACT

Model-driven engineering is significant in modern software development, as it leverages models to guide software design and implementation. However, legacy systems are characterized by outdated architectures and poor documentation, which pose challenges for maintenance. Model-Driven Reverse Engineering (MDRE) addresses these issues by extracting models from legacy systems, enabling better comprehension. This study explores clustering techniques to extract high-level concepts from source code. Clustering reveals system concepts that can be transformed into models by grouping similar code entities. These models provide a foundation for system modernization and further development. Unlike existing MDRE approaches, this study systematically evaluates multiple clustering techniques and preprocessing scenarios to assess their ability to recover high-level system concepts from legacy source code. The results highlight the impact of preprocessing and noise on concept extraction, providing insights for applying clustering within model-driven modernization workflows.

Keywords-clustering; machine learning; model-driven architecture; model-driven engineering; source code

I. INTRODUCTION

In today's software development landscape, the complexity and scale of systems continue to grow, making it increasingly difficult to manage and maintain existing codebases [1]. Model-driven Engineering (MDE) has emerged as an alternative approach to address this challenge by shifting the focus from code-centric development to model-based abstraction. Its success lies in reducing manual coding efforts and maintaining consistency between design and implementation, making it a preferred approach for modern software systems [2]. The Model-Driven Architecture (MDA) initiative proposed by the Object Management Group (OMG) provides a well-established framework for implementing MDE [3]. MDA proposes a structured approach based on a set of models, including the Platform-Independent Model (PIM), the Platform-Specific Model (PSM), and the Platform Description Model (PDM).

As systems evolve, there is a growing need to extend MDE principles to existing legacy systems that implement outdated architectures and lack proper documentation. MDRE addresses this need by enabling the recovery of high-level models from legacy systems [4]. The primary goal of MDRE is to extract

conceptual information from the system's artifacts and represent them in models such as the Unified Modeling Language (UML) diagrams. These recovered models facilitate the understanding, maintenance, and modernization of legacy systems, ensuring adaptability to new requirements. A key challenge in MDRE is recovering meaningful artifacts from large and poorly documented codebases. Clustering techniques have proven to be highly effective in addressing this challenge by grouping source code artifacts and identifying the architectural concepts within a system [5]. Techniques such as k-means can be employed to analyze and organize source code. Clustering allows for the identification of high-level concepts by grouping related classes, methods, or modules based on their shared characteristics.

The present study proposes an approach that leverages clustering techniques as part of the MDRE process to recover high-level concepts from legacy systems. Building on previous work where traditional grouping analyses were used to group similar source code artifacts [6]. It also explores clustering algorithms to enhance and simplify the process. By evaluating the effectiveness of several clustering methods in extracting conceptual information from source code, the proposed approach provides a methodology that supports the modernization of legacy systems within the MDE field. The initial results are promising, particularly when using k-means and agglomerative clustering techniques alongside preprocessing strategies tailored specifically to the characteristics of source code.

II. RELATED WORK

Source code clustering is an essential technique in software engineering, particularly for enhancing software comprehension, maintenance, and concept identification [7]. An important contribution to semantic clustering was made in [8], where a technique based on LSI and clustering was proposed to group source code artifacts with similar vocabulary [5]. Extending clustering applications in software engineering, authors in [9] introduced a methodology for extracting information from C# code by combining clustering with association rules mining. Clone detection is another area where clustering is applied. Authors in [10] modified a clone detection tool based on Locality-Sensitive Hashing (LSH) by replacing the LSH algorithm with k-means clustering.

A broader and up-to-date overview of the MDRE techniques and challenges has been provided, including the integration of machine learning and language models [4]. The use of large language models for reverse engineering tasks, such as the extraction of OCL constraints from legacy systems, has also been explored [11].

III. METHODOLOGY

A. Model-Driven Modernization

1) Metamodel

This study defines a set of metamodels to describe the core constructs of the proposed modernization approach. The foundation of this approach and metamodel is the notion of concepts, to identify and extract these concepts from a system using clustering techniques. Figure 1 illustrates a simplified representation of the metamodel of this approach. It highlights the core modeling concepts and their roles in supporting the extraction of domain and platform-level abstractions during reverse engineering. The base metamodel acts as the foundational construct of the approach. It also defines the essential elements of a software system, including its core concepts and associated constraints. The approach focuses on extracting domain-specific and platform-specific concepts from the source code of a legacy system using this metamodel. Complementing this, the PDM metamodel describes the platform or architecture of a given system. It comprises two components: a UML profile and a set of transformations. The UML profile is a customizable extension of the UML metamodel that allows designers to adapt UML to specific domains or platforms. The transformations metamodel defines how transformations are specified and combined to develop advanced systems as part of the forward engineering process. A detailed description of the metamodel is provided in [12].

2) Method Overview

Figure 2 shows the overview of the proposed model-driven modernization approach, designed to migrate a legacy system to a modern one. The process is divided into two main phases: reverse engineering [6] and forward engineering. The reverse engineering phase involves extracting models from the legacy system's source code, including the PSM, PIM, and PDM. During refactoring, users can manipulate these models to add, remove, or modify concepts. In the forward engineering phase, the refined PIM and a new PDM are used to generate the source code for the modernized system.

This work focuses on the reverse engineering phase, analyzing the legacy system's source code to extract concepts through clustering techniques. In [6], a hypothesis was proposed suggesting that code related to the implementation platform tends to exhibit repetitive or semi-repetitive patterns across the system's source code. To validate this hypothesis, clone detection and LSI techniques were employed to group classifiers and operations with shared code and vocabulary. Building on this foundation, this study experiments with clustering techniques to group related classifiers, offering an alternative methodology for identifying concepts in legacy systems.

B. Clustering

Grouping classifiers (i.e., program abstractions such as classes and interfaces) within a system is a critical step in the modernization approach, as it directly impacts the outcome of the discovery process. Incorrectly identified groups at the beginning can lead to misclassification of concepts. Thus, an experiment aimed at leveraging clustering techniques was conducted. Figure 3 presents the proposed concept identification methodology using clustering. The pipeline from dataset preparation and preprocessing to feature extraction and clustering-based concept identification is summarized.

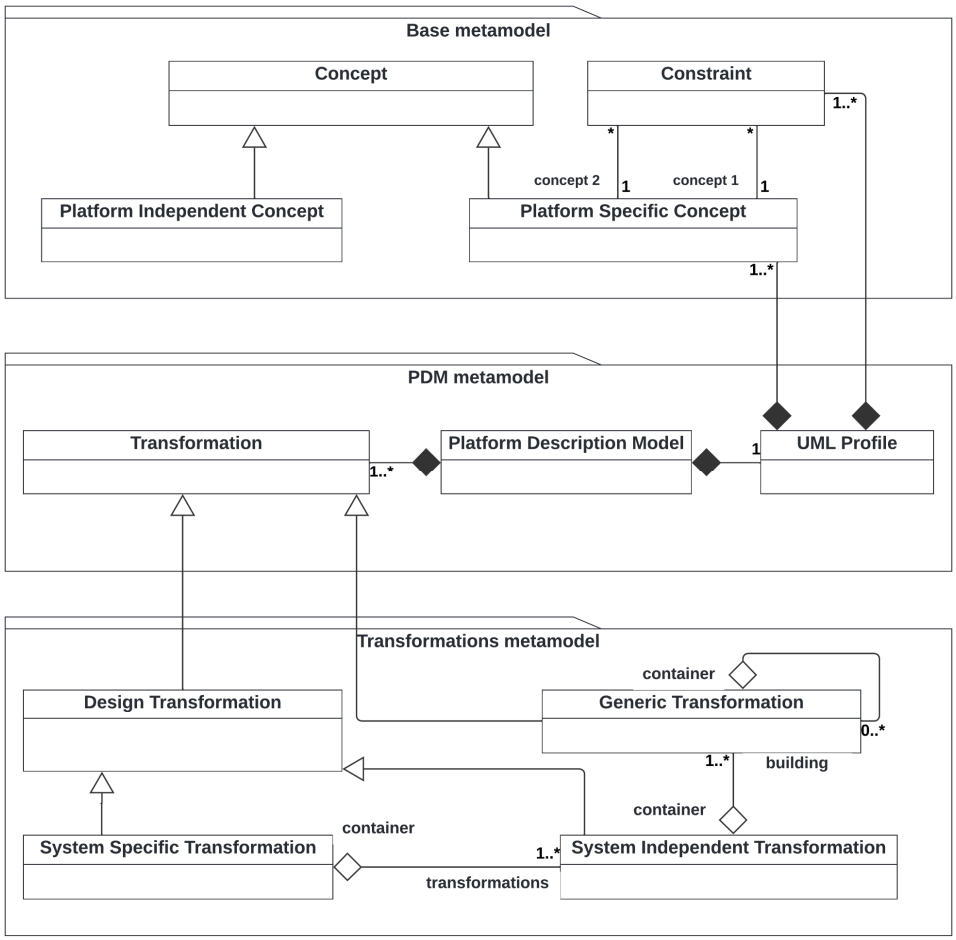

Fig. 1. Simplified version of the metamodel of the proposed approach.

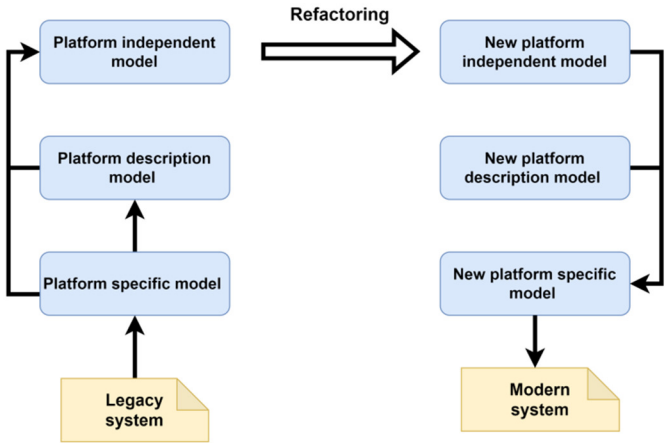

Fig. 2. Overview of model-driven modernization approach.

The initial step includes preparation of the dataset, which consists of source code files from systems developed in C# within the .NET framework. To build the dataset, open-source systems on GitHub were searched with specific criteria. The second step is pre-processing. It is a critical step, serving as the foundation for subsequent techniques like clustering and classification.

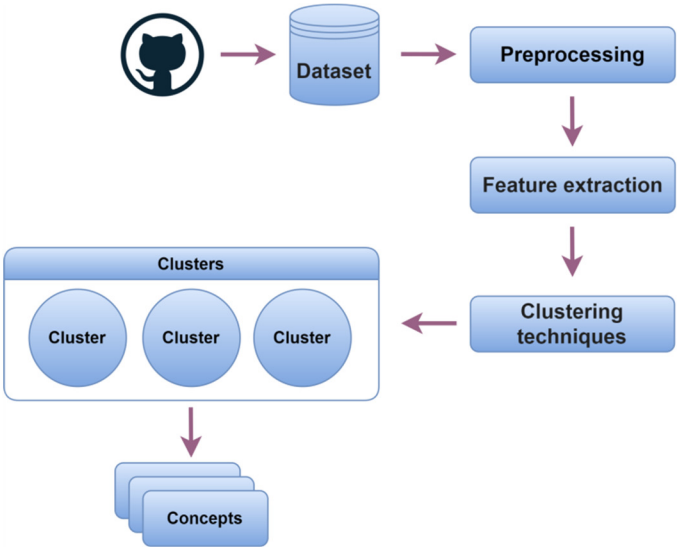

Fig. 3. Source code clustering methodology.

It prepares textual data for analysis through tokenization, normalization, handling of comments and string literals, and stop-word removal. The third step is feature extraction, an

important process in data analysis that involves identifying and representing the most relevant attributes from raw data [13]. Several approaches are employed, such as word frequency (TF-IDF) [14], word embeddings (Word2Vec) [15], and pre-trained models (BERT) [16]. Finally, clustering techniques are applied to group objects, ensuring that those within the same group share greater similarity than those in different groups. These methods fall into various categories such as partitioning methods.

IV. IMPLEMENTATION

A. Dataset

Software projects from GitHub were chosen as the dataset, ensuring that they fulfill specific criteria to support the analysis. All projects were developed in C#, with a minimum of 2,000 LOC and at least 50 classifiers. The dataset includes systems that were last updated over five years ago. Additionally, each project was licensed under terms permitting analysis, such as the LGPL and MIT licenses. The selection process resulted in four systems, as presented in Table I: CLRZMQ [17], GhostRunner [18], RoslynSecurityGuard [19], and TeamCitySharp [20].

Although validated on C# source code, the approach is largely language-agnostic and would require only minor preprocessing adaptations, mainly the removal of language-specific keywords, to support other systems.

TABLE I. SELECTED GITHUB PROJECTS AS DATASET

Software system	Size		Last update (years)	License
	LOC	Classifier number		
CLRZMQ	9593	107	12	LGPL
GhostRunner	6055	143	9	MIT
RoslynSecurityGuard	9461	73	7	LGPL
TeamCitySharp	3068	87	8	MIT

B. Pre-Processing

Source code preprocessing methods are used to refine the dataset for more effective concept discovery. The initial preprocessing involves merging partial classes if they exist within the source code to remove any potential variation in class representation. Next, programming language vocabulary, such as keywords, control structures, and loop constructs, was removed. These elements do not contribute to the conceptual understanding of the problem and solution domains. To further enhance the discovery process, identifiers and variable names were segregated based on their naming conventions, such as snake_case and PascalCase. Naming conventions often carry semantic meaning, reflecting the purpose of the variable or identifier.

C. Vectorization

TF-IDF was employed to capture the most relevant characteristics of the source code. Several other techniques are commonly used for this purpose, including Word2Vec and GloVe. For source code clustering, TF-IDF is often a better choice than neural embeddings such as Word2Vec. This is because source code contains fewer semantic relationships between words compared to natural language documents, and its structure relies more on syntax and specific token frequencies. TF-IDF efficiently captures these distinctions without requiring large-scale contextual understanding, providing accurate representations for source code, where variables, class names, and functions often carry unique meanings based on frequency rather than semantic similarity.

D. Clustering

Various clustering techniques, including k-means, agglomerative clustering, BIRCH, DBSCAN, and Spectral clustering, were used to assess their effectiveness in grouping similar code elements. The elbow and Silhouette methods [21] were applied to estimate the number of clusters. The elbow method was inconclusive due to the absence of a clear inflection point. Consequently, the Silhouette method, which evaluates both cohesion and separation, was used. This technique proved more suitable for capturing structural similarities in code artifacts.

When clustering techniques are applied to source code, the output is a set of clusters grouping classifiers. At this stage, the objective is not to directly extract detailed structural elements, such as attributes or relationships, but to identify coherent groups representing high-level system concepts. These clusters serve as an abstraction layer that guides subsequent reverse engineering activities, where detailed attributes, operations, and relationships are recovered through model-driven transformations and expert validation. This separation allows clustering to reduce system complexity while preserving sufficient information to support re-engineering tasks.

V. VALIDATION

A. Validation Framework

1) Validation Setup

In the validation phase of the reverse engineering approach, a series of experiments were conducted to evaluate the effectiveness of the methodology under various configurations and settings. Figure 4 illustrates the validation process, which incorporates multiple datasets, preprocessing scenarios, clustering techniques, and evaluation metrics to ensure comprehensive validation. The experiments utilized four datasets and explored three distinct preprocessing scenarios. These preprocessing scenarios correspond to different methods of cleaning and preparing source code text before applying clustering, to analyze their impact on the clustering task:

- The without preprocessing (WoPP) scenario applies no preprocessing.
- The preprocessing with stemming (PPwS) scenario includes tokenization, stop-word removal, stemming, and source-code-specific preprocessing such as keyword removal and identifier splitting.
- The preprocessing without stemming (PPWoS) scenario applies the same preprocessing steps but excludes stemming to isolate its impact on clustering performance.

To further test the robustness of this approach, it was validated using both normal and noisy datasets. The normal

datasets consisted of clean source code collected from GitHub, while the noisy datasets were derived by introducing modifications, such as character deletion and swapping, to simulate noise. Furthermore, multiple evaluation metrics were employed to measure clustering performance, capturing various dimensions of clustering quality. A Python desktop application [22] was developed to facilitate the evaluation process.

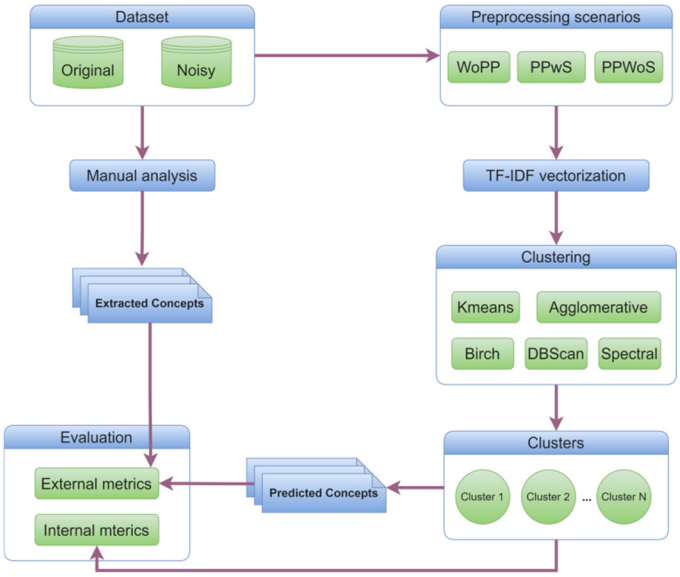

Fig. 4. Validation process of source code clustering.

2) Metrics

To evaluate the proposed clustering approach, both internal and external evaluation metrics were employed. Internal metrics assess the quality of clusters based on the data themselves without reference to external labels. These metrics measure properties such as cluster cohesion and separation. The internal metrics include Silhouette score [23], Calinski-Harabasz index [24], and Davies-Bouldin index [25]. The Silhouette score ranges from −1 to 1, where higher values indicate better-defined clusters with stronger intra-cluster cohesion and inter-cluster separation. Values around 0.2–0.3, as observed in the present study, indicate moderate but meaningful structure, which is common in high-dimensional source code data. Similarly, a higher Calinski-Harabasz index reflects better cluster separation; although absolute values are dataset-dependent, relative comparisons across scenarios and techniques remain informative.

External metrics require predefined labels to compare the clusters' output with a known classification. These metrics measure the alignment of the clustering results with the true classes. To perform this external evaluation, a manual analysis of the system's source code was conducted to identify the concepts implemented within the system, which serve as the ground truth. Using the ground truth and predicted concepts, the precision, recall, and F-score metrics were calculated.

VI. RESULTS AND DISCUSSION

The clustering experiment results are divided into internal and external metrics across three scenarios: WoPP, PPwS, and PPWoS, each reflecting performance under differing noise conditions.

A. Internal Evaluation

1) WoPP Scenario

Clustering techniques showed considerable performance variation. In the original dataset, k-means achieved Silhouette scores of 0.226 for CLRZMQ and 0.218 for GhostRunner, with lower values for RoslynSecurityGuard and TeamCitySharp, and substantial declines in noisy datasets. Calinski-Harabasz index remained low across systems, with only moderate k-means performance for GhostRunner in the original dataset, but the index declined under noise. Davies-Bouldin index showed that BIRCH and DBSCAN struggled with compactness in noisy datasets, while agglomerative clustering performed slightly better for GhostRunner and RoslynSecurityGuard, but not for TeamCitySharp. Spectral clustering showed a high Silhouette score for CLRZMQ originally, declining sharply with noise.

2) PPwS Scenario

PPwS improved cohesion, as k-means demonstrated improved Silhouette scores for CLRZMQ, and the Calinski-Harabasz index increased, reaching 6.777 for CLRZMQ and 9.110 for GhostRunner. DBSCAN consistently failed in a noisy dataset, while Spectral clustering achieved the highest Silhouette score (0.294) for CLRZMQ and demonstrated strong performance across systems.

3) PPWoS Scenario

PPWoS performed slightly worse than PPwS but better than WoPP. For CLRZMQ, k-means, and agglomerative clustering reached Silhouette scores of 0.273 and 0.278 in the original dataset. Davies-Bouldin index remained high in noisy datasets, with k-means reaching 2.667 for GhostRunner. Spectral clustering scored 0.259 for CLRZMQ but declined with noise. Calinski-Harabasz indices were higher in the original datasets across all methods.

B. External Evaluation

1) WoPP Scenario

For external metrics, k-means achieved a precision of 73.33% for GhostRunner in the original dataset, while precision declined to 60.0% with the noisy dataset. Recall was strong for RoslynSecurityGuard (78.57%) but declined for TeamCitySharp in the noisy dataset (11.11%). F-score values followed these trends, declining from 64.71 to 19.05. Agglomerative clustering performed better under noise, with a precision score of 63.64% for GhostRunner, but had a low recall value of 35.71%. BIRCH and DBSCAN showed poor results in noise, failing to produce meaningful clusters.

2) PPwS Scenario

WoPP resulted in improved performance, achieving a k-means precision score of 83.33% for GhostRunner, and maintained a strong F-score of 68.75% for the noisy dataset. Agglomerative clustering remained strong, with a precision score of 63.16% for TeamCitySharp, while demonstrating a consistent performance. Spectral clustering performed well with a noisy dataset, with F-score of 58.06% for CLRZMQ and 46.15% for GhostRunner, whereas DBSCAN continued to lack robustness under noise.

3) PPWoS Scenario

The results showed similar trends, with k-means achieving a precision score of 76.92% for CLRZMQ under the original dataset, which decreased to 66.67% under noise. F-score also declined, with TeamCitySharp dropping from 64.86% to 19.05. Agglomerative clustering maintained robust performance for the noisy dataset, with 66.67% F-score for CLRZMQ. DBSCAN and BIRCH performed poorly, especially with noise.

C. Discussion

1) Impact of Preprocessing and Noise

A key objective of the present study is to demonstrate the effect of preprocessing, especially the effect of stemming, on clustering performance. PPwS results consistently surpassed WoPP and PPWoS across different methods and datasets, especially in noisy environments. PPwS reduced noise by consolidating similar terms, producing tighter clusters [26]. k-means and agglomerative clustering showed noticeable improvements in Silhouette scores and Calinski-Harabasz indices in PPwS, indicating better cohesion and clearer separation. PPWoS marginally improved performance over WoPP, suggesting that stemming reduces dimensionality and redundancy, allowing algorithms to focus on high-level concepts rather than terminology variations. Noise posed major challenges for BIRCH and DBSCAN, whose Davies-Bouldin index increased sharply, indicating reduced compactness and separation. In noisy scenarios, DBSCAN often produced only a single cluster. k-means and agglomerative clustering were more resilient but also degraded under noise. These findings confirm that abstraction and noise reduction play an important role in effective concept recovery, and further extend this observation by providing a quantitative comparison of clustering techniques under controlled noise conditions.

2) Clustering Techniques Performance

k-means achieved the highest Silhouette scores across scenarios, forming well-defined clusters due to its centroid-based nature and resilience in PPwS, allowing the extraction of meaningful concepts despite data complexity. Agglomerative clustering also performed strongly, particularly on larger datasets like TeamCitySharp, achieving higher Silhouette scores and Calinski-Harabasz indices. Its hierarchical approach enabled fine-grained cluster formation, beneficial for diverse systems. Spectral clustering showed promise when conceptual similarities were subtle, performing relatively well in PPwS due to eigenvalue-based detection, but was more sensitive to noise, with reduced metrics in noisy datasets. BIRCH and DBSCAN performed poorly in noisy datasets, frequently failing to return meaningful clusters.

VII. CONCLUSION

This study presented an approach to source code clustering aimed at extracting high-level concepts to support Model-Driven Reverse Engineering (MDRE) tasks. By employing various clustering techniques, the study demonstrated the potential of these methods to uncover concepts from source code, even in legacy systems with limited documentation.

Preprocessing, particularly stemming, emerged as a critical factor in improving clustering outcomes. The results showed that clustering techniques, especially k-means and agglomerative clustering, performed significantly better in the preprocessing with stemming (PPwS) scenario. This underscores the importance of linguistic normalization in reducing noise and improving the extraction of high-level concepts. In this sense, k-means and agglomerative clustering consistently outperformed other methods. Their ability to handle noisy data and still produce meaningful clusters makes them well-suited for concept extraction in software systems. In contrast, techniques such as BIRCH and DBSCAN were highly sensitive to noise, often failing to return useful clusters in noisy scenarios.

Future research should explore hybrid clustering approaches or methods specifically designed to handle noisy datasets, such as fuzzy clustering [27]. Additionally, techniques such as ensemble clustering [28], where the results of multiple clustering algorithms are combined, could improve robustness and accuracy.

DATA AVAILABILITY STATEMENT

Data supporting the findings of this study is available at: https://github.com/AHamza14/Source-code-clustering.

REFERENCES

[1] M. W. Godfrey and D. M. German, "The Past, Present, and Future of Software Evolution," in *2008 Frontiers of Software Maintenance*, Beijing, China, Sept. 2008, pp. 129–138, https://doi.org/10.1109/FOSM.2008.4659256.

[2] C. Verbruggen and M. Snoeck, "Practitioners' Experiences with Model-Driven Engineering: A Meta-Review," *Software and Systems Modeling*, vol. 22, no. 1, pp. 111–129, Feb. 2023, https://doi.org/10.1007/s10270-022-01020-1.

[3] J. Miller and J. Mukerji, *MDA Guide Version 1.0.1,* Needham, MA, USA: Object Management Group, 2003.

[4] H. A. Siala, K. Lano, and H. Alfraihi, "Model-Driven Approaches for Reverse Engineering—A Systematic Literature Review," *IEEE Access*, vol. 12, pp. 62558–62580, 2024, https://doi.org/10.1109/ACCESS.2024.3394732.

[5] A. Kuhn, S. Ducasse, and T. Gîrba, "Semantic Clustering: Identifying Topics in Source Code," *Information and Software Technology*, vol. 49, no. 3, pp. 230–243, Mar. 2007, https://doi.org/10.1016/j.infsof.2006.10.017.

[6] H. Abdelmalek, G. Chénard, I. Khriss, and A. Jakimi, "A Bimodal Approach for the Discovery of a View of the Implementation Platform of Legacy Object-Oriented Systems under Modernization Process," in *Proceedings of 35th International Conference on Computers and Their Applications*, pp. 83–93, Mar. 2020, https://doi.org/10.29007/rbp7.

[7] Q. I. Sarhan, B. S. Ahmed, M. Bures, and K. Z. Zamli, "Software Module Clustering: An In-Depth Literature Analysis," *IEEE*

Transactions on Software Engineering, vol. 48, no. 6, pp. 1905–1928, Jun. 2022, https://doi.org/10.1109/TSE.2020.3042553.

[8] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman, "Indexing by Latent Semantic Analysis," *Journal of the American Society for Information Science*, vol. 41, no. 6, pp. 391–407, Sept. 1990, https://doi.org/10.1002/(SICI)1097-4571(199009)41:6%253C391::AID-ASI1%253E3.0.CO;2-9.

[9] Y. Kanellopoulos, C. Makris, and C. Tjortjis, "An Improved Methodology on Information Distillation by Mining Program Source Code," *Data & Knowledge Engineering*, vol. 61, no. 2, pp. 359–383, May 2007, https://doi.org/10.1016/j.datak.2006.06.002.

[10] A. Ashish, "Clones Clustering Using K-means," in *10th International Conference on Intelligent Systems and Control (ISCO)*, Coimbatore, India, Jan. 2016, pp. 1–6, https://doi.org/10.1109/ISCO.2016.7726943.

[11] H. A. Siala and K. Lano, "Towards Using LLMs in the Reverse Engineering of Software Systems to Object Constraint Language," in *2025 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, Montreal, QC, Canada, Mar. 2025, pp. 1–6, https://doi.org/10.1109/SANER64311.2025.00096.

[12] H. Abdelmalek, I. Khriss, and A. Jakimi, "Towards an Effective Approach for Composition of Model Transformations," *Frontiers in Computer Science*, vol. 6, Jun. 2024, Art. no. 1357845, https://doi.org/10.3389/fcomp.2024.1357845.

[13] A. O. Salau and S. Jain, "Feature Extraction: A Survey of the Types, Techniques, Applications," in *2019 International Conference on Signal Processing and Communication (ICSC)*, NOIDA, India, Mar. 2019, pp. 158–164, https://doi.org/10.1109/ICSC45622.2019.8938371.

[14] S. Qaiser and R. Ali, "Text Mining: Use of TF-IDF to Examine the Relevance of Words to Documents," *International Journal of Computer Applications*, vol. 181, no. 1, pp. 25–29, Jul. 2018, https://doi.org/10.5120/ijca2018917395.

[15] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient Estimation of Word Representations in Vector Space." arXiv, 2013, https://doi.org/10.48550/ARXIV.1301.3781.

[16] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding." arXiv, 2018, https://doi.org/10.48550/ARXIV.1810.04805.

[17] ZeroMQ, "CLRZMQ." GitHub, 2011, [Online]. Available: https://github.com/zeromq/clrzmq.

[18] G. Marland, "GhostRunner." GitHub, 2014, [Online]. Available: https://github.com/gmarland/GhostRunner.

[19] D. S. Guard, "RoslynSecurityGuard." GitHub, 2017, [Online]. Available: https://github.com/dotnet-security-guard/roslyn-security-guard.

[20] P. Stack, "TeamCitySharp." GitHub, 2016, [Online]. Available: https://github.com/stack72/TeamCitySharp.

[21] T. M. Kodinariya, "Review on Determining Number of Cluster in K-Means Clustering," *International Journal of Advance Research in Computer Science and Management Studies*, vol. 1, no. 6, Jan. 2013. pp. 90–95.

[22] H. Abdelmalek, "Source Code Clustering." GitHub, 2024, [Online]. Available: https://github.com/AHamza14/Source-code-clustering.

[23] P. J. Rousseeuw, "Silhouettes: A Graphical Aid to the Interpretation and Validation of Cluster Analysis," *Journal of Computational and Applied Mathematics*, vol. 20, pp. 53–65, Nov. 1987, https://doi.org/10.1016/0377-0427(87)90125-7.

[24] T. Calinski and J. Harabasz, "A Dendrite Method for Cluster Analysis," *Communications in Statistics - Theory and Methods*, vol. 3, no. 1, pp. 1–27, 1974, https://doi.org/10.1080/03610927408827101.

[25] D. L. Davies and D. W. Bouldin, "A Cluster Separation Measure," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-1, no. 2, pp. 224–227, Apr. 1979, https://doi.org/10.1109/TPAMI.1979.4766909.

[26] M. H. Ahmed, S. Tiun, N. Omar, and N. S. Sani, "Short Text Clustering Algorithms, Application and Challenges: A Survey," *Applied Sciences*, vol. 13, no. 1, Dec. 2022, Art. no. 342, https://doi.org/10.3390/app13010342.

[27] S. Askari, N. Montazerin, and M. H. Fazel Zarandi, "Generalized Possibilistic Fuzzy C-Means with Novel Cluster Validity Indices for Clustering Noisy Data," *Applied Soft Computing*, vol. 53, pp. 262–283, Apr. 2017, https://doi.org/10.1016/j.asoc.2016.12.049.

[28] K. Golalipour, E. Akbari, S. S. Hamidi, M. Lee, and R. Enayatifar, "From Clustering-to-Clustering Ensemble Selection: A Review," *Engineering Applications of Artificial Intelligence*, vol. 104, Sept. 2021, Art. no. 104388, https://doi.org/10.1016/j.engappai.2021.104388.