Date of publication JUN-30, 2025, date of current version MAR-16, 2025.
www.computingonline.net / computing@computingonline.net

Print ISSN 1727-6209
Online ISSN 2312-5381
DOI 10.47839/ijc.24.2.4024

Methodology for Determining the Optimal Clustering Algorithm for Software Quality Verification

VLADYSLAV PARASHCHENKO, OLEH BEREST
Computer Sciences Department, Sumy State University, Sumy, Ukraine

Corresponding author: Vladyslav Parashchenko (e-mail: vladyslav.parashchenko@student.sumdu.edu.ua)

ABSTRACT The article examines methodologies for evaluating the quality of clustering algorithms used to identify patterns within codebases in the context of a decision support system (DSS) module for software quality verification in information and communication systems. A novel feature dictionary is introduced, wherein evaluation metrics represent a software class as an implementation vector. These metrics are preselected based on the most salient characteristics of programming code. The five widely recognized clustering algorithms - namely, K-Means, DBSCAN, OPTICS, Affinity Propagation, and Gaussian Mixture Models are evaluated in this study. The proposed methodology is applied to five Java application projects that implement diverse architectural solutions and software patterns. These applications are distributed under an open license and are readily accessible for research purposes. The source code of the selected software is transformed into vectors by extracting relevant code characteristics, thereby facilitating subsequent training. The results obtained confirmed the suitability of the proposed feature vector, and the optimal clustering model was subsequently selected for integration into the decision-making system module for quality assessment in information and communication systems.

KEYWORDS software quality; software metrics; clustering; K-Means; DBSCAN; OPTICS; Affinity Propagation; Gaussian Mixture; DSS.

I. INTRODUCTION

Nowadays, there are billions of software systems. Ensuring their correct functionality necessitates continuous updates, maintenance, and enhancements. Furthermore, the integration of a developer into an existing project presents significant challenges, as it requires not only the implementation of functional components but also adherence to the stylistic and structural conventions of the existing codebase. To address this issue, the implementation of a decision support system capable of analyzing code and assessing the compatibility of new functionalities with the established framework would be highly advantageous. Such a system would substantially expedite both development and verification processes by enabling experts to concentrate on substantive modifications to the software system, rather than dedicating time to rectifying stylistic discrepancies and preserving the overall architectural integrity of the system.

A variety of static analysis tools exist to address these issues, enabling the identification of vulnerabilities, rectification of errors, and monitoring of predefined metrics. The premise that program code represents the sole definitive source of knowledge is sound, as alternative representations - such as diagrams or specifications - only partially capture the correctness of new functionality. In addition to conventional static analysis methods, it is advisable to incorporate data mining techniques to extract supplementary factors and discern architectural patterns within the system under development. These techniques facilitate the derivation of rules that can be applied to refine coding styles during the development process. Given the frequent turnover within development teams [1] and the inherent challenges associated with knowledge transfer [2], there arises a critical need to convert internal empirical rules into verifiable standards. According to the Broken Windows Theory [3], even minor deviations from established norms may ultimately precipitate systemic disorder; applied to software development, this suggests that the accumulation of low-quality code exacerbates maintenance complexity.

Consequently, the primary objective is to investigate methodologies for verifying the quality of software within information and communication systems, with the aim of formalizing these rules. This approach would ensure greater consistency, maintainability, and scalability of software systems in the face of evolving team dynamics and project requirements.

II. MATERIALS AND METHODS

Clustering represents one of the most widely utilized methods in the field of data mining [4]. To identify patterns within data, it is essential to represent the data in a format that is computationally interpretable and to select an appropriate algorithm. Clustering falls under the category of unsupervised machine learning tasks, making it particularly well-suited for the analysis of program code. This is due to the fact that it does not require pre-processing of a labeled training dataset, and the patterns identified are derived directly from the existing codebase. The outcomes of clustering enable the identification of similar objects within the codebase, facilitate their grouping into coherent categories, and support the maintenance of a consistent stylistic approach across the implementation of diverse functionalities. This method thus serves as a valuable tool for enhancing code uniformity and reducing variability in software development practices.

As the subject of investigation, codebases utilizing the object-oriented programming language Java are selected. Given that the atomic unit of code in Java is the class, the study focuses on exploring the potential for identifying patterns among distinct classes through the application of clustering techniques. This approach allows for the systematic analysis of structural and functional similarities within the codebase, leveraging the inherent organization of Java's class-based architecture.

As part of this research, five widely recognized algorithms frequently employed in clustering tasks are selected for evaluation:

- K-Means Algorithm [5, 6]. This is one of the most prominent methods in clustering, where the assignment of data points to clusters is determined by minimizing the root-mean-square distance to the centroid of each cluster. A significant limitation of this approach is the necessity to predefine the number of clusters. Additionally, its computational complexity of O(n2) [7] makes it less efficient for large datasets.
- DBSCAN Algorithm [8]. This algorithm partitions data based on density, requiring two parameters for optimization: the minimum distance between points within a cluster and the minimum number of elements required to form a cluster. Unlike K-Means, DBSCAN autonomously determines the number of clusters, making it advantageous when the expected number of clusters is unknown. Furthermore, enhancements to the DBSCAN algorithm were proposed to reduce computational time for high-dimensional datasets [9].
- OPTICS Algorithm [10, 11]. Similar to DBSCAN, OPTICS operates on the principle of density but is optimized to automatically identify the minimum distance between points. While this optimization simplifies practical application, it increases the computational complexity of the algorithm.
- Gaussian Mixture Algorithm [12, 13]. Unlike K-Means, this algorithm is capable of identifying clusters in more complex data structures. It achieves this through an iterative process that estimates the parameters of a normal distribution for each cluster, allowing for greater flexibility in modeling data.
- Affinity Propagation Algorithm [14]. This is one of the most advanced clustering techniques, relying on the exchange of "messages" between data points to determine cluster assignments. A key advantage of this method is that it does not require the number of clusters to be specified in advance; instead, it identifies the optimal number of clusters during the iterative process.

These algorithms were chosen for their diverse approaches to clustering, each offering unique strengths and limitations, which were evaluated in the context of software code analysis.

Since the clustering task is a supervised learning task, the quality of clustering can only be checked using metrics that evaluate the separation result. The most well-known metric for evaluating the quality of clustering is silhouette [15], which estimates that the elements of one cluster are more similar than the elements of two different clusters. The metric score ranges from -1 to 1 depending on the quality of cluster separation. Metric (3) is calculated as the ratio of the average distance (a_i) between points within the same cluster (1) to the ratio of the average distance between points in different clusters b_i (2).

$$a_i = \frac{1}{|C_i| - 1} \sum_{j \in C_{i,}\ i \neq j} d_{ij}, \tag{1}$$

where C_i - count of points in cluster; $\underline{d_{ij}}$ - distance between points.

$$b_i = \min_{j \neq i} \frac{1}{|C_j|} \sum_{j \in C_{j,}\ i \neq j} d_{ij}, \tag{2}$$

where C_j – count of points in other cluster; $\underline{d_{ij}}$ - distance between points.

$$s = \begin{cases} 1 - \frac{a_i}{b_i}, & if\ a_i < b_i \\ 0, & if\ a_i = b_i. \\ \frac{a_i}{b_i} - 1, & if\ a_i > b_i \end{cases} \tag{3}$$

As an alternative to the silhouette metric, there is the Davies–Bouldin score (5) [16], which evaluates the ratio of the variance of points in one cluster to the ratio of the variance in different clusters (4).

$$R_{ij} = \frac{S_i + S_j}{d_{ij}} \text{,} \tag{4}$$

where S_i – average distance between points and center of cluster i; S_j – average distance between points and center of cluster j.

$$DB = \frac{1}{k} \sum_{i=1}^{k} \max R_i \text{,} \tag{5}$$

where k – count of clusters; R_i – separation level in cluster i.

The third approach to evaluating clustering results is the Calinski-Harabasz index (8). This metric is computed using the formulas presented in equations (6-7). A low index value indicates that the clusters are widely dispersed, whereas a well-separated clustering structure should yield a high index value.

$$BCSS = \sum_{i=1}^{k} n_i \left||c_i - c|\right| \quad ,(6)$$

where n_i – number of points in cluster; c_i – center of cluster i; c – general center of data cluster.

$$\text{WCSS} = \sum_{i=1}^{k} \sum_{x \in Ci} \left||x - c_i|\right| \quad ,(7)$$

where k – count of clusters; x – point in cluster; c_i – center of cluster i.

$$CH = \frac{BCSS}{\text{WCSS}} \quad ,(8)$$

where BCSS – weighted sum of squared Euclidean distance between center and general center of data cluster; WCSS – squared Euclidean distance between point and their cluster centers.

Most clustering methodologies necessitate the specification of the number of expected clusters to evaluate their performance. However, there is no universally accepted approach to addressing this challenge. In study [18], the authors evaluated the quality of the resulting partitions through cross-validation, employing various models in conjunction with cluster assessment metrics. Meanwhile, other researchers proposed modified algorithms that incorporate optimization steps directly within the clustering process [19, 20]. These approaches aim to enhance the accuracy and efficiency of clustering by integrating iterative refinement mechanisms into the algorithmic framework.

An alternative to predefining the number of clusters is to adopt an empirical approach. With comprehensive knowledge of the underlying data structure, one can infer the expected number of clusters, subsequently optimizing the results based on these estimates. For instance, employing the pure architecture approach proposed by R. Martin [21] suggests that the anticipated number of clusters should approximate the number of application layers.

In a typical Java web application, one would expect to find a controller class characterized by low cyclomatic complexity that serves as the entry point for processing HTTP requests. However, controller classes often exhibit low cohesion, as they typically delegate control to other classes. In contrast, entity classes - which represent domain objects and correspond to relational database tables - usually contain numerous properties, a limited number of methods with low cyclomatic complexity, and similarly display low cohesion. Repository classes, which abstract access to the data store, tend to have higher cyclomatic complexity, increased cohesion, and fewer properties. Service classes, responsible for implementing business logic, generally exhibit high cyclomatic complexity and, provided that developers adhere to the principle of single responsibility, high cohesion.

A similar architectural paradigm is observed in mobile applications, where models are responsible for data management. Given that SQLite is the predominant relational database solution in mobile environments, the repository and entity pattern remains equally pertinent. Moreover, the construction of user interfaces relies on specialized view classes, which are characterized by both high cyclomatic complexity and high cohesion. Considering the above analysis, the following metrics are chosen to construct the representation vectors that should characterize the classes:

Table 1. Evaluated class metrics

Metrics	Description
Cyclomatic complexity	metric of cyclomatic complexity of a class;
NPath complexity	another metric of cyclomatic complexity
Cognitive complexity	metric of cognitive complexity of code
Boolean expression complexity	a metric of the complexity of Boolean expressions
Class length	class size metric, equal to the number of significant strings in the class
File length	class size metric, total file size
Average method length	class size metric, total file size
Average line length	string length metric, which is equal to the average string length
Class FanOut complexity	metric of the number of classes used in this class
Average If depth	metric for nesting conditional statements;
Average For depth	nesting metric for loop statements
Average Try depth	nesting metric for try...catch blocks
Law of Demeter	metric of violation of Demeter's law, the number of violations in a separate class
LCOM1[22]	lack of class cohesion metric(LCOM1)
LCOM2[22]	lack of class cohesion metric(LCOM2)
LCOM3[22]	lack of class cohesion metric(LCOM3)
LCOM4[22]	lack of class cohesion metric(LCOM4)
LCOM5[22]	lack of class cohesion metric(LCOM5)

The article proposes an investigation into the efficacy of a generated representative feature vector for partitioning a codebase into clusters, followed by an evaluation of the results using predefined quality metrics. This approach aims to assess the suitability of the feature vector in facilitating meaningful and accurate clustering within the context of software code analysis.

III. RELATED WORKS

Among the existing body of research on this topic, it is noteworthy to highlight study [23], in which the authors employed clustering techniques to identify similar code fragments. The primary objective of this work is to develop a tool for detecting plagiarism; however, the methodology proposed can be extended to other related tasks. A similar approach was adopted in research [24], where hierarchical clustering was utilized to detect analogous code segments. The authors evaluated their results using precision, recall, and F-measure metrics to validate the effectiveness of their approach.

In another study [25], the authors explored the relationships between software metrics by applying the K-Means clustering

technique. Research [26] further advanced this methodology by enhancing the performance of the K-Means algorithm. Specifically, the authors implemented a parallelized version of K-Means and introduced additional data transformations to reduce the dimensionality of input vectors, thereby improving computational efficiency.

Research [27] also contributed to this domain by investigating various methods of code representation. The authors constructed models that leverage abstract syntax trees, control flow graphs, and bytecode as sources of knowledge for deep learning models. Their findings demonstrate that models incorporating multiple code representations excel in tasks such as classification and code clone detection. Similarly, the authors of [28] proposed a novel model for generating embeddings based on the BERT architecture. Their approach demonstrated strong performance in code clustering and labeling tasks, further underscoring the potential of advanced representation techniques in software analysis.

Collectively, these studies highlight the diverse applications of clustering and representation methods in software engineering, offering valuable insights into their effectiveness for tasks ranging from plagiarism detection to code analysis and classification.

IV. RESULTS

To assess the effectiveness of the proposed methodology for identifying similar elements within a codebase using selected feature vectors, a sample of five representative application projects was selected. Given the widespread use of the Java programming language in both server-side and mobile application development, the chosen projects encompass systems that employ diverse architectural paradigms. These include:

- Spring REST [29]. This project serves as an example of a standard application that provides a RESTful API for various purposes, adhering to conventional Java API development practices.
- Spring GraphQL [30]. This API is developed using GraphQL technology as an alternative to REST, offering a server equipped with the necessary set of mutations and queries for data manipulation.
- Java EE Application [31]. This application is constructed using technologies provided by the Java EE platform, representing a traditional enterprise-level architecture.
- Android MVVM (Model-View-ViewModel) [32]. This architecture exemplifies a common pattern for Android application development, emphasizing the separation of view and model components to improve testability and maintainability.
- Android MVP (Model-View-Presenter) [33]. Similar to MVVM, this architecture is widely used in Android development, further promoting separation of concerns and modular design.

A more detailed overview of the selected projects, including their specific characteristics and metrics, is provided in Table 2. This selection ensures a comprehensive evaluation of the proposed methodology across a range of architectural styles and application domains.

Table 2. Software selected for analysis

Project	Spring GraphQL	Spring REST	JiavaEE	Android MVVM	Android MVP
Number of classes	93	50	122	60	116
Number of rows	3800	2500	6200	7000	8254

These projects were analyzed and divided into sets of possible components presented in Table 3.

Table 3. Components descriptions

Project	Name of Component	Description	Count of components
Spring Graph QL	Resolver	Entry point for query or mutation	8
	Repository	Abstraction over database to data access	
	Entity	Mapping between database object and business logic object	
	Configurations	Framework setup classes	
	Exceptions	Objects to represent program errors	
	Interfaces	Public contact for classes	
	Utils	Collection of static methods that perform common tasks.	
	DTO	Plain java objects to exchange data between different systems	
Spring REST	Controller	Entry point for http request	9
	Services	Business logic component	
	Repository	Abstraction over database to data access	
	Entity	Mapping between database object and business logic object	
	Configurations	Framework setup classes	
	Exceptions	Objects to represent program errors	
	Interfaces	Public contact for classes	
	Utils	Classes that perform common tasks.	
	DTO	Plain java objects to exchange data between different systems	
Java EE	Resources	Entry point for request	9
	Service	Business logic component	
	Repository	Abstraction over database to data access	
	Entity	Mapping between database object and business logic object	
	Configurations	Framework setup classes	
	Exceptions	Objects to represent program errors	
	Interfaces	Public contact for classes	
	Utils	Classes that perform common tasks.	
	DTO	Plain java objects to exchange data between different systems	

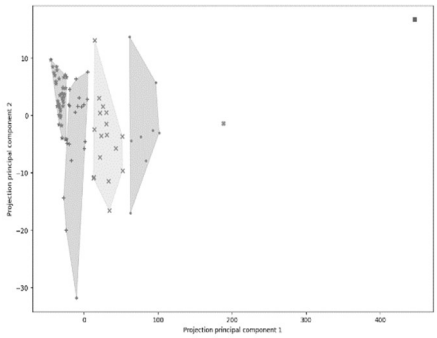
Android MVVM	View	Activity that represents logic to display data for users	10
	Model	Business logic components that encapsulate data store	
	ViewModel	Components that contain logic to handle events from view and changes from data store	
	Controller	Logic to manage several view models	
	Repository	Abstraction over database to data access	
	Entity	Mapping between database object and business logic object	
	Configurations	Framework setup classes	
	Interfaces	Public contact for classes	
	Utils	Classes that perform common tasks.	
	Broadcast receivers	Components to listen events from operation system and other apps	
Android MVP	View	Usually, activity that represents logic to display data for users	9
	Model	Business logic components that encapsulate data store	
	Presenter	Class that represents data for view	
	Repository	Abstraction over database to data access	
	Entity	Mapping between database object and business logic object	
	Configurations	Framework setup classes	
	Interfaces	Public contact for classes	
	Utils	Classes that perform common tasks.	
	Broadcast receivers	Components to listen events from operation system and other apps	

The source code of each project was subjected to analysis, during which metrics for the classes enumerated in Table 1 were computed. This process resulted in the generation of input vectors, the quantity of which corresponds to the number of classes within each project. Subsequently, the clustering procedure was executed utilizing the selected algorithms, and the corresponding quality metrics were calculated. The outcomes of this analysis are presented in Table 4.

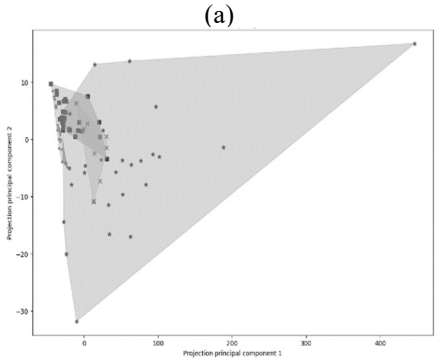
The initial dataset cannot be effectively visualized in a two-dimensional space; therefore, the Principal Component Analysis (PCA) technique [34] was employed to illustrate the clustering results. In PCA, the axes are referred to as principal components (PC1, PC2, etc.), with each component representing a direction in the data space that captures the maximum variance within the dataset. Figure 1 depicts the clustering outcomes for a web application developed using the Spring framework and GraphQL technology. As evident from the graphical representation, the K-Means model yielded the most favorable results, while the Affinity Propagation and Gaussian models also exhibited high cluster resolution. In contrast, the DBSCAN model failed to achieve a clear separation of clusters within the given dataset, resulting in overlapping clusters, as illustrated in Figure 1. This observation is corroborated by the silhouette metric for the DBSCAN model, which returned a negative value, indicating suboptimal clustering performance.

Table 4. Software selected for analysis

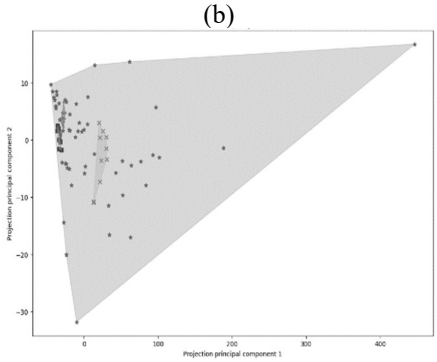
	Project	Spring GraphQL	Spring REST	Java EE	Android MVVM	Android MVP
KMeans	Silhouette metric	0.56	0.46	0.59	0.64	0.58
	Davis-Baldwin metric	0.49	0.63	0.63	0.47	0.57
	Kalinski-Harabash metric	311	73	200	1114	329
	Number of classes	6	7	9	9	9
DBSCAN	Silhouette metric	-0.22	0.4	0.57	0.46	-0.01
	Davis-Baldwin metric	1.25	0.93	1.41	1.52	1.54
	Kalinski-Harabash metric	3.04	8.09	29	16.1	5.3
	Number of classes	4	2	3	3	5
OPTICS	Silhouette metric	0.39	0.12	0.11	-0.14	0.02
	Davis-Baldwin metric	1.21	1.41	1.15	1.5	1.28
	Kalinski-Harabash metric	22	20	13	1.84	15
	Number of classes	4	6	9	6	13
Affinity Propagation	Silhouette metric	0.1	0.44	0.41	0.36	0.4
	Davis-Baldwin metric	1.34	0.53	0.60	0.51	0.4
	Kalinski-Harabash metric	1014	74	284	1951	324
	Number of classes	9	9	7	12	13
Gaussian Mixture	Silhouette metric	0.33	0.36	0.49	0.5	0.49
	Davis-Baldwin metric	0.87	0.65	0.63	0.52	0.46
	Kalinski-Harabash metric	298	73	264	1341	255
	Number of classes	6	8	9	8	7



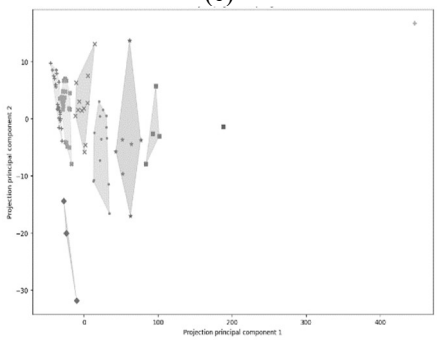
(a)



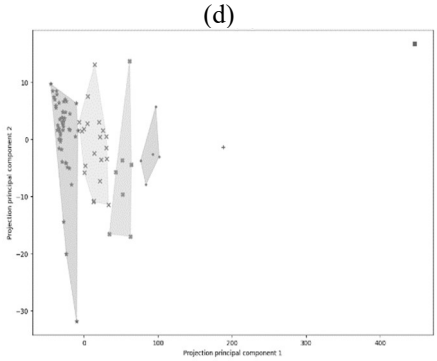
(b)



(c)



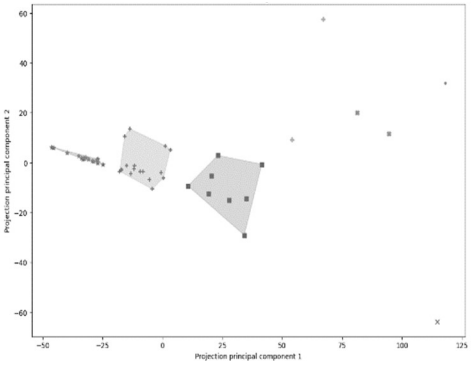
(d)



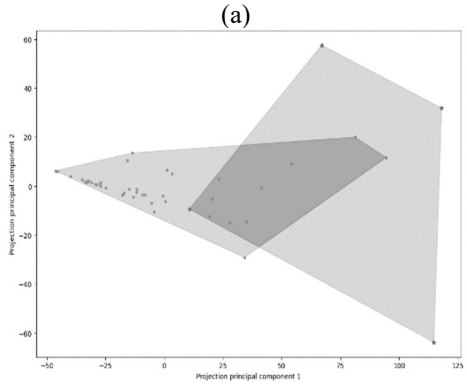
(e)

Figure 1. Visualization for the code base of the GraphQL application based on the Spring framework
a) K-Means model, b) DBSCAN model, c) OPTICS model, d) Affinity propagation model, e) Gaussian Mixture model

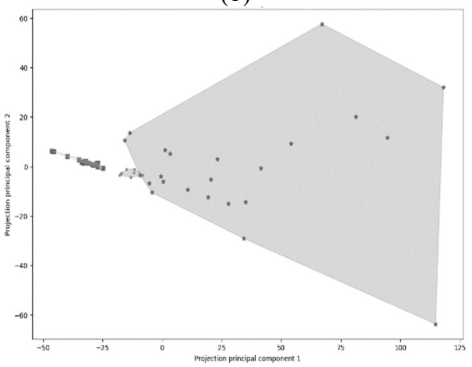
Figure 2 presents the results of clustering applied to a web application built on the Spring framework, which implements a REST API. As illustrated in the graphs, the DBSCAN and OPTICS models demonstrated relatively poor clustering performance, despite achieving higher metric scores compared to the previous application. Notably, the silhouette metric score for the DBSCAN model exceeded that of the Gaussian Mixture model, although it underperformed in other metrics. This discrepancy can be attributed to the elongated and irregular shapes of the clusters, which deviate from the rounded forms typically associated with more effective clustering outcomes.



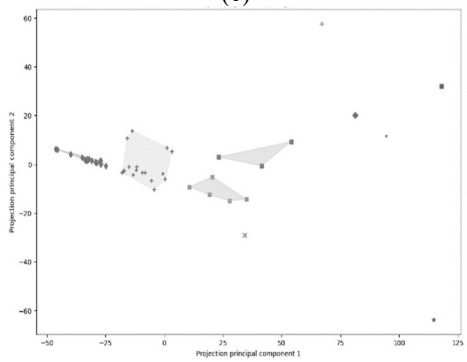
(a)



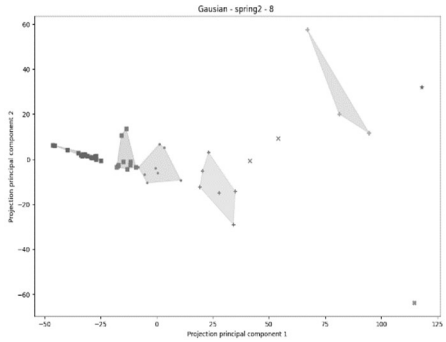
(b)



(c)



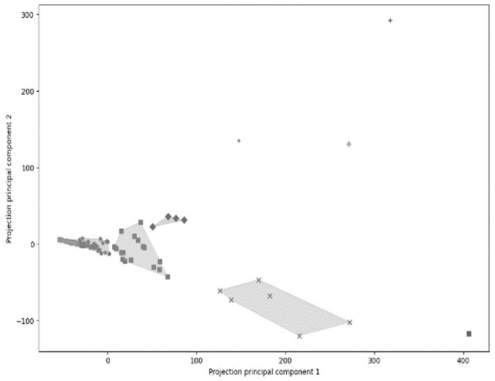
(d)



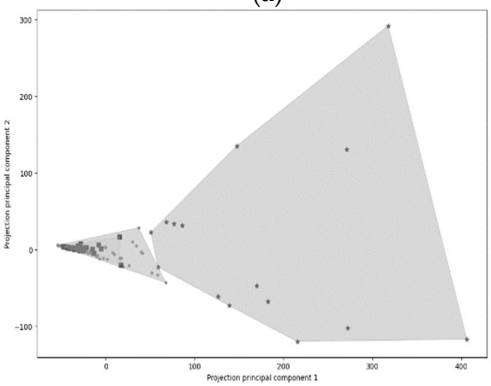
(e)

Figure 2. Visualization for the REST API code base of the application using the Spring framework a) KMeans model, b) DBSCAN model, c) OPTICS model, d) Affinity propagation model, e) Gaussian Mixture model

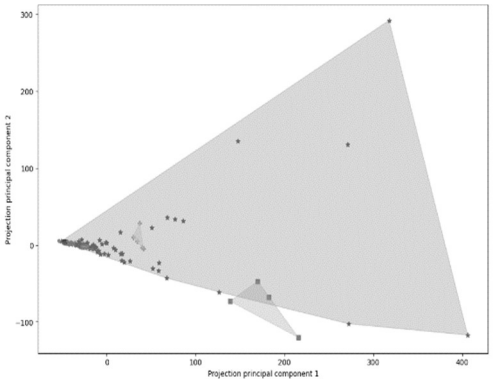
According to Figure 3, the DBSCAN and OPTICS models once again demonstrated suboptimal performance when applied to the web application project based on JavaEE technology. While the silhouette metric for these models appears relatively high, the Davis-Bouldin and Calinski-Harabasz metrics reveal the inadequacy of such clustering outcomes. Specifically, the ratio of the Calinski-Harabasz index for the DBSCAN model to that of the Affinity Propagation model is 9.79, further underscoring the inferiority of the DBSCAN model's clustering performance in this context.



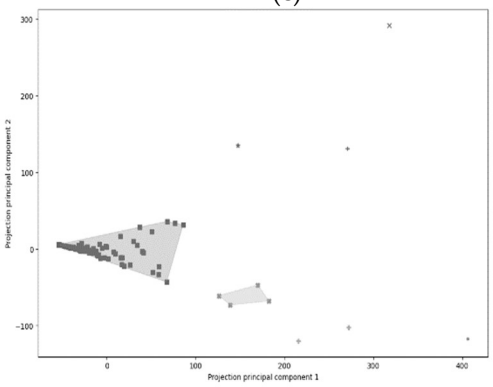
(a)



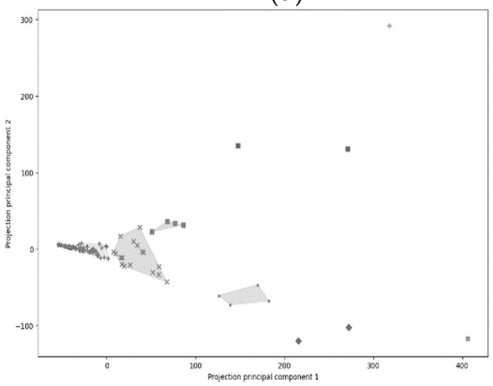
(b)



(c)



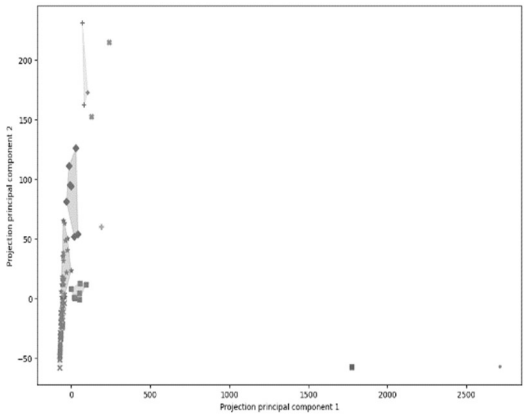
(d)



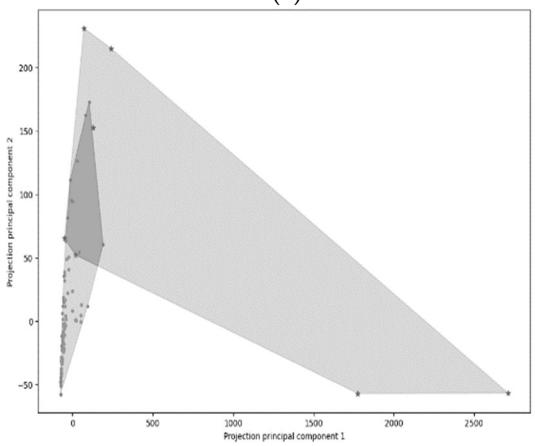
(e)

Figure 3. Visualization for the code base of the application built on JavaEE technology
a) KMeans model, b) DBSCAN model, c) OPTICS model, d) Affinity propagation model, e) Gaussian Mixture model

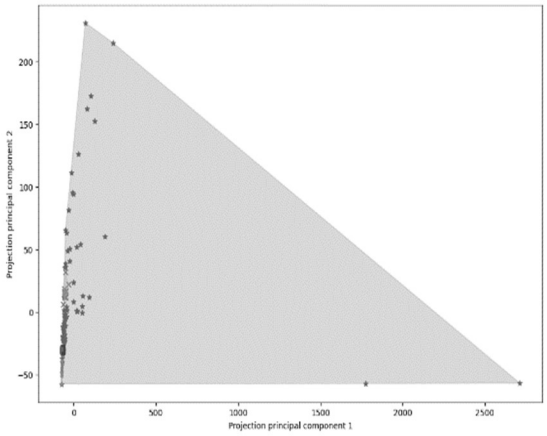
Figure 4 illustrates the visual representation of clustering results for a mobile application utilizing the MVVM architecture. As observed in Figures 4.a and 4.d, the models have achieved a relatively effective separation of clusters, supported by high metric scores. Notably, the Calinski-Harabasz score for the Affinity Propagation model exceeds that of the K-Means model, despite underperforming in the other two metrics. Conversely, the silhouette score for the OPTICS model falls below zero, a result visually corroborated in Figure 4.c, where a single large cluster dominates, encompassing the majority of data points. This poor separation is further validated by the other quality metrics, which collectively indicate the inadequacy of the OPTICS model in this context.



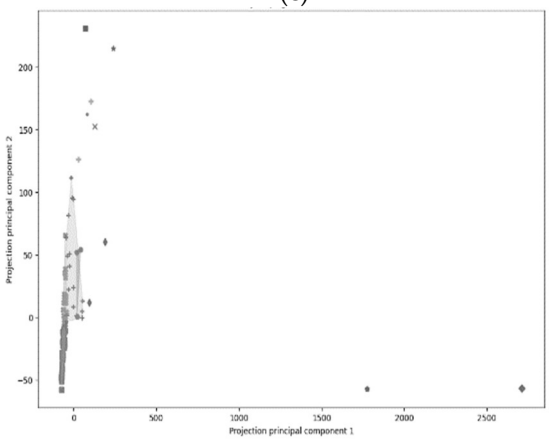
(a)



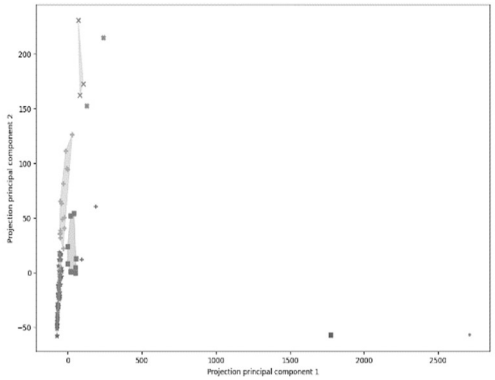
(b)



(c)



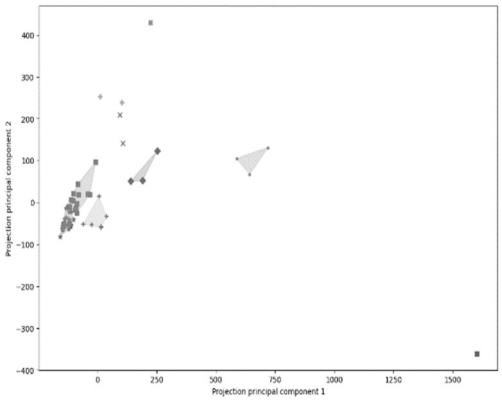
(d)



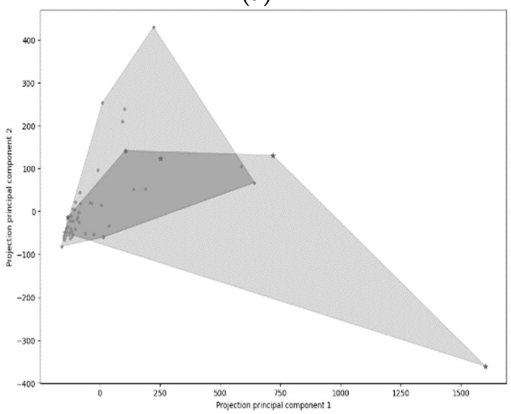
(e)

Figure 4. Visualization for the code base of a mobile application using MVVM architecture a) KMeans model, b) DBSCAN model, c) OPTICS model, d) Affinity propagation model, e) Gaussian Mixture model

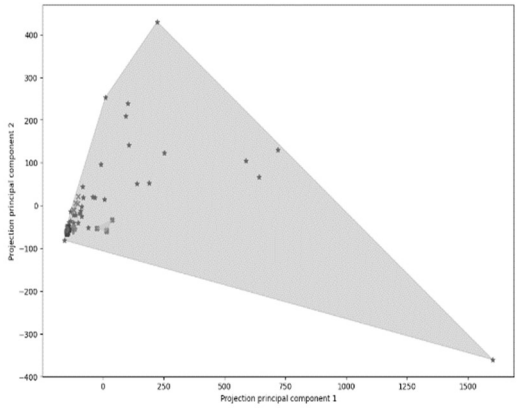
Figure 5 presents the clustering outcomes for a mobile application employing the MVP architecture. For the DBSCAN model, the silhouette score is negative, the Calinski-Harabasz metric yields a low value, and the Davies-Bouldin metric, in contrast, is significantly high. The ratio of the Davies-Bouldin metric results between the K-Means and DBSCAN models is approximately 3, favoring the K-Means model. Conversely, the ratio of the scores between the K-Means and Affinity Propagation models is nearly 0, indicating that the clustering quality of these two models is comparable. This suggests that while DBSCAN underperforms, K-Means and Affinity Propagation demonstrate similar effectiveness in this context.



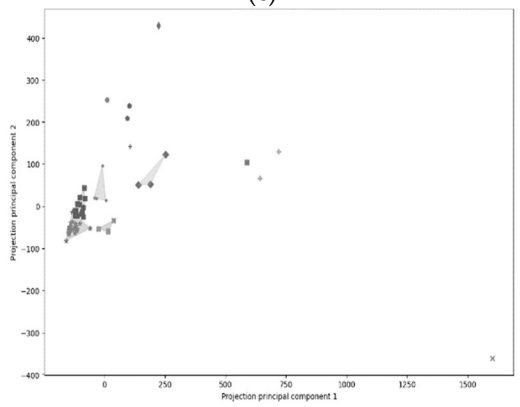
(a)



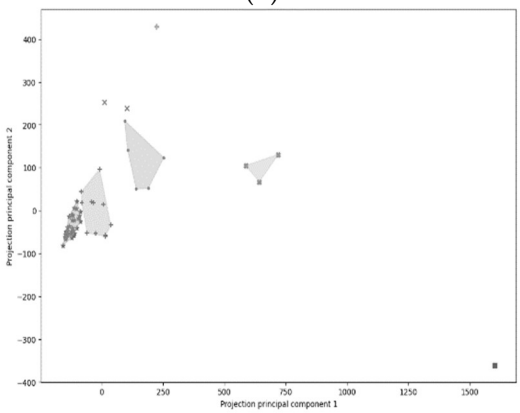
(b)



(c)



(d)



(e)

Figure 5. Visualization for the code base of a mobile application with MVP architecture
a) KMeans model, b) DBSCAN model, c) OPTICS model, d) Affinity propagation model, e) Gaussian Mixture model

Table 3 presents the clustering results along with the corresponding values of the metrics used to evaluate clustering quality. Based on the findings, the K-Means model achieved high-quality cluster separation, as evidenced by its consistently high scores across all metrics. This performance was further corroborated by additional visualizations of the clustering outcomes. The number of clusters obtained through metric-based optimization closely aligns with the empirically expected number of clusters, reinforcing the model's effectiveness.

In contrast, the DBSCAN model exhibited poor performance in the majority of experiments. Although it achieved a favorable silhouette metric score for the JavaEE web application, the Calinski-Harabasz and Davies-Bouldin metrics indicate suboptimal separation quality. Additionally, the number of clusters generated by DBSCAN was significantly lower than the expected number. Similarly, the OPTICS model underperformed in most experiments, with its most inaccurate results observed for a mobile application based on the MVVM architecture.

The Affinity Propagation model, however, demonstrated high-quality results across all cases examined. The average differences between its metrics and those of the K-Means model were -0.224 for the silhouette metric, 0.118 for the Davies-Bouldin metric, and 324 for the Calinski-Harabasz metric, indicating that its performance is comparable to that of K-Means. A notable advantage of the Affinity Propagation model is its ease of use, as it does not require prior analytical determination of the number of clusters, eliminating the need for extensive pre-analysis of input data. In most instances, the number of clusters produced by this model exceeded expectations, suggesting a tendency to identify outliers. This characteristic could prove beneficial for detecting anomalous software classes.

The Gaussian Mixture model also yielded positive results, with average differences of -0.132, 0.068, and 40.8 compared to K-Means for the silhouette, Davies-Bouldin, and Calinski-Harabasz metrics, respectively. Visualization further confirmed the high quality of cluster separation achieved by this model. Overall, these findings highlight the strengths and limitations of each clustering approach in the context of software class analysis.

V. CONCLUSIONS

The demonstrated results confirm the feasibility of employing clustering techniques to identify patterns in the structure and construction of software code. Among the models evaluated, the K-Means, Affinity Propagation, and Gaussian Mixture models emerged as the most suitable, exhibiting strong performance in both visual representation and established clustering metrics. Notably, the Affinity Propagation model stands out as the most appropriate choice due to its ease of use and high-quality clustering outcomes for codebases.

The scientific contribution of this study lies in the development of a novel methodology for assessing codebase clustering and its application to verify the quality of software in information and communication systems. Central to this methodology is the use of a specialized feature vector, uniquely designed to represent each class, which captures the functional purpose and characteristics of software classes.

From a practical perspective, this research introduces a new approach for evaluating the effectiveness of codebase clustering and identifying patterns in software construction. This approach is highly versatile, as it leverages existing static analysis tools to derive the necessary metrics. Furthermore, the proposed methodology for processing and visualizing clustering results is applicable to any object-oriented programming language. This framework is intended to serve as the foundation for a quality control module within a Decision Support System (DSS) for verifying software in information and communication systems.

Future research will focus on refining the feature vector used to characterize software classes for clustering purposes. Additionally, the demonstrated technology will be instrumental in developing decision rules for the DSS, enhancing its capability to ensure software quality and maintainability.

References

[1] R. E. S. Santos, F. Q. B. da Silva, M. T. Baldassarre, and C. V. C. de Magalhães, "Benefits and limitations of project-to-project job rotation in software organizations: A synthesis of evidence," *Inf Softw Technol*, vol. 89, pp. 78–96, 2017. https://doi.org/10.1016/j.infsof.2017.04.006.

[2] P. Silva de Garcia, M. Oliveira, and K. Brohman, "Knowledge sharing, hiding and hoarding: how are they related?" *Knowledge Management Research & Practice*, vol. 20, no. 3, pp. 339–351, 2022. https://doi.org/10.1080/14778238.2020.1774434.

[3] D. Thomas and A. Hunt, *The pragmatic programmer*, Addison-Wesley Professional, 2019.

[4] A. A. Yahya, A. Osman, "Using data mining techniques to guide academic programs design and assessment," *Procedia Computer Science*, vol. 163, pp. 472-481, 2019. https://doi.org/10.1016/j.procs.2019.12.130.

[5] J. A. Hartigan and M. A. Wong, "A k-means clustering algorithm," *Appl Stat*, vol. 28, no. 1, pp. 100–108, 1979. http://dx.doi.org/10.2307/2346830.

[6] B. Karthikeyan, D. J. George, G. Manikandan, and T. Thomas, "A comparative study on K-means clustering and agglomerative hierarchical clustering," *International Journal of Emerging Trends in Engineering Research*, vol. 8, no. 5, pp. 1600-1604, 2020. http://dx.doi.org/10.30534/ijeter/2020/20852020.

[7] M. Ahmed, R. Seraj, and S. M. S. Islam, "The k-means algorithm: A comprehensive survey and performance evaluation," *Electronics* (Basel), vol. 9, no. 8, p. 1295, 2020. https://doi.org/10.3390/electronics9081295.

[8] R. S. V Chandrasekar and G. A. Britto, "Comprehensive review on density-based clustering algorithm in data mining," *Int J Res Anal*, vol. 6, no. 2, pp. 5–9, 2019.

[9] S. Weng, J. Gou, and Z. Fan, "h-DBSCAN: A simple fast DBSCAN algorithm for big data," *Proceedings of the Asian Conference on Machine Learning, PMLR*, 2021, pp. 81–96.

[10] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander, "OPTICS: Ordering points to identify the clustering structure," *ACM Sigmod Record*, vol. 28, no. 2, pp. 49–60, 1999. http://dx.doi.org/10.1145/304182.304187.

[11] Z. Deng, Y. Hu, M. Zhu, X. Huang, and B. Du, "A scalable and fast OPTICS for clustering trajectory big data," *Cluster Comput*, vol. 18, pp. 549–562, 2015. http://dx.doi.org/10.1007/s10586-014-0413-9.

[12] C. A. Bouman, M. Shapiro, G. W. Cook, C. B. Atkins, and H. Cheng, "Cluster: An unsupervised algorithm for modeling Gaussian mixtures." 1997. [Online]. Available at: https://engineering.purdue.edu/~bouman/software/cluster/manual.pdf.

[13] Y. Zhang et al., "Gaussian mixture model clustering with incomplete data," *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, vol. 17, no. 1s, pp. 1–14, 2021. https://doi.org/10.1145/3408318.

[14] B. J. Frey and D. Dueck, "Clustering by passing messages between data points," *Science*, vol. 315, no. 5814, pp. 972–976, 2007. https://doi.org/10.1126/science.1136800.

[15] K. R. Shahapure and C. Nicholas, "Cluster quality analysis using silhouette score," *Proceedings of the 2020 IEEE 7th international conference on data science and advanced analytics (DSAA)*, 2020, pp. 747–748. https://doi.org/10.1109/DSAA49011.2020.00096.

[16] D. L. Davies and D. W. Bouldin, "A cluster separation measure," *IEEE Trans Pattern Anal Mach Intell*, no. 2, pp. 224–227, 1979. https://doi.org/10.1109/TPAMI.1979.4766909.

[17] T. Caliński, and J. Harabasz, "A dendrite method for cluster analysis," *Communications in Statistics*, vol. 3, no. 1, pp. 1–27, 1974. https://doi.org/10.1080/03610927408827101.

[18] A. Vysala and D. J. Gomes, "Evaluating and validating cluster results," *Proceedings of the 9th International Conference on Data Mining & Knowledge Management Process (CDKP'2020)*, 2020, https://doi.org/10.5121/csit.2020.100904.

[19] A. Shafeeq and K. S. Hareesha, "Dynamic clustering of data with modified k-means algorithm," *Proceedings of the 2012 Conference on Information and Computer Networks*, 2012, pp. 221–225. http://dx.doi.org/10.13140/2.1.4972.3840.

[20] M. A. Masud, M. M. Rahman, S. Bhadra, and S. Saha, "Improved k-means algorithm using density estimation," *Proceedings of the 2019 IEEE International Conference on Sustainable Technologies for Industry 4.0 (STI)*, 2019, pp. 1–6. https://doi.org/10.1109/STI47673.2019.9068033.

[21] R. C. Martin, *Clean architecture*, Prentice Hall, 2017.

[22] E. N. H. Kirgil та T. E. Ayyildiz, "Analysis of lack of cohesion in methods (LCOM): A case study," *Proceedings of the 2021 2nd IEEE Int. Inform. Softw. Eng. Conf. (IISEC)*, Ankara, Turkey, 16–17 December 2021, pp. 1-4. https://doi.org/10.1109/IISEC54230.2021.9672419.

[23] M. Ďuračík, E. Kršák, and P. Hrkút, "Searching source code fragments using incremental clustering," *Concurr Comput*, vol. 32, no. 13, p. e5416, 2020. https://doi.org/10.1002/cpe.5416.

[24] Y. Amaliah, W. Musu, and M. Fadlan, "Auto clustering source code to detect plagiarism of student programming assignments in Java programming language," *Proceedings of the 2021 3rd IEEE International Conference on Cybernetics and Intelligent System (ICORIS)*, 2021, pp. 1–6. https://doi.org/10.1109/ICORIS52787.2021.9649465.

[25] B. Mathur and M. Kaushik, "In object-oriented software framework improving maintenance exercises through k-means clustering approach," *Proceedings of the 2018 3rd IEEE International Conference on Internet of Things: Smart Innovation and Usages (IoT-SIU)*, 2018, pp. 1–7. https://doi.org/10.1109/IoT-SIU.2018.8519897.

[26] P. Hrkút, M. Ďuračík, M. Mikušová, M. Callejas-Cuervo, and J. Zukowska, "Increasing K-means clustering algorithm effectivity for using in source code plagiarism detection," *Proceedings of the International Conference on Smart Technologies, Systems and Applications*, Springer, 2019, pp. 120–131.

[27] M. Tufano, C. Watson, G. Bavota, M. Di Penta, M. White, and D. Poshyvanyk, "Deep learning similarities from different representations of source code," *Proceedings of the 15th International Conference on Mining Software Repositories*, 2018, pp. 542–553. https://doi.org/10.1145/3196398.3196431.

[28] M. Hägglund, F. J. Pena, S. Pashami, A. Al-Shishtawy, and A. H. Payberah, "Coclubert: Clustering machine learning source code," *Proceedings of the 2021 20th IEEE International Conference on Machine Learning and Applications (ICMLA)*, 2021, pp. 151–158. https://doi.org/10.1109/ICMLA52953.2021.00031.

[29] E. Ozdemir, "A general overview of RESTful web services," *Advances in Systems Analysis, Software Engineering, and High Performance Computing*. *IGI Glob.*, 2020, pp. 133–165. https://doi.org/10.4018/978-1-7998-2142-7.ch006.

[30] P. Mandani, Lolith Raj B. K., Nithyananda R. Shetty and Rahul T. N., "A comprehensive analysis of GraphQL," SSRN Electron. J., 2024. https://doi.org/10.2139/ssrn.4915678.

[31] J. Juneau, RESTful Web Services. In: Java EE 8 Recipes. Apress, Berkeley, CA. https://doi.org/10.1007/978-1-4842-3594-2_15.

[32] N. S. P. K. Yadati, "Architecture Design (MVVM + Clean Architecture)," *J. Artif. Intell., Mach. Learn. Data Sci.*, vol. 1, no. 3, pp. 703–706, 2023. https://doi.org/10.51219/JAIMLD/naga-satya-praveen-kumar-yadati/177.

[33] R. F. García, "MVP: Model–View–Presenter," iOS Architecture Patterns, Berkeley, CA: Apress, 2023, pp. 107–144. https://doi.org/10.1007/978-1-4842-9069-9_3.

[34] M. Greenacre, P. J. F. Groenen, T. Hastie, A. I. D'Enza, A. Markos and E. Tuzhilina, "Principal component analysis," *Nature Rev. Methods Primers*, vol. 2, no. 1, 2022. https://doi.org/10.1038/s43586-022-00184-w.

***Vladyslav Parashchenko** is a PhD student in Sumy State University. Previously received a bachelor and master's degree in Computer Sciences in Sumy State University. Research interests: in software development, software architecture, development automatization*

***Oleh Berest**, Software engineer in Swisscom LTD. Research interests: software development, machine learning, pattern recognition, multidimensional visualization.*