Performance Evaluation of Data Mining Techniques to Enhance the Reusability of Object-Oriented (O-O) Systems

Bharti Bisht
Manav Rachna International Institute of Research and Studies (MRIIRS), India
onebharti@gmail.com (corresponding author)

Parul Gandhi
Manav Rachna International Institute of Research and Studies (MRIIRS), India
parul.fca@mriu.edu.in

Received: 8 March 2024 | Revised: 1 April 2024, 19 April 2024, and 29 April 2024 | Accepted: 15 May 2024

Licensed under a CC-BY 4.0 license | Copyright (c) by the authors | DOI: https://doi.org/10.48084/etasr.7213

ABSTRACT

The software industry is evolving at a rapid pace, making it necessary to optimize efforts and accelerate the software development process. Software can be reused to achieve quality and productivity goals. Reusability is a crucial measure that can be used to increase the overall level of software quality in less time and effort. To better understand the necessity of enhancing the software reusability of Object-Oriented (O-O) systems, this study employed a semi-automated approach to measure the values of class-level software metrics on an input dataset collected from the MAVEN repository. This paper explored several previous studies, data strategies, and tools to predict reusability in O-O software systems. This study compares various data mining techniques to identify the most suitable approach for enhancing the reusability of O-O software systems. The analysis was based on performance parameters such as precision, MSE, and accuracy rates. Due to its higher precision and lower MSE, the SOM technique is considered one of the top data mining approaches to increase the reusability of O-O software systems. However, the results show that the different levels of reusability in O-O software systems are not adequately addressed in current solutions.

Keywords-object-oriented (O-O) system; data mining; reusability level; product quality

I. INTRODUCTION

In a society that relies heavily on software and is increasingly susceptible to software failures, software professionals are under tremendous pressure to create more complex and advanced software systems. With the increase in the number of computer professionals, the development of technologies that improve hardware and computing performance, and massive increases in memory and storage, the growth in the expected size of the software systems required is exponential. With increasing demands for even more sophisticated systems and maintenance of current software, it has become difficult to maintain the increase in software productivity. After decades of extensive research in software engineering, software reuse has emerged as the only practical and technically possible way to achieve the desired increases in productivity and quality that the software industry demands. The reusability of a software component serves as a key indicator of its capacity to be reused. Components can be considered autonomous, replaceable parts of applications carrying explicit and distinct functions. The reusability concept extends beyond the source code, including design, documentation, and test cases. Reusing the actual information from real software data, such as project status and progress, requires more sophisticated techniques. Software components (functions and processes) can be tested in traditional procedural programming before being saved in libraries for later use. This is certainly a source for reuse, but these components are restricted to essential application areas. It is difficult to reuse the software components of normal environments, except for mathematics subroutines, because there is a very low chance that they perfectly correspond to the requirements of any subsequent application. However, modification is challenging and might occasionally be more expensive than redesign if the software components are not understood. On the other hand, in Object-Oriented (O-O) programming, adaptation is achieved by extending or specializing preexisting components rather than by altering them. O-O programming languages allow software to be designed in various forms, including components, making it easy for users to reuse massive amounts of code. Increasing the reusability levels requires more emphasis on implementing O-O methodologies and employing an effective strategy to achieve this objective. Previous studies have shown that there are O-O components that can help fix software bugs and produce higher-quality software [1]. Object orientation is emphasized in the O-O software creation process in several phases, including evaluation, design, and deployment.

A key element in the success of software reuse is the ability to predict the needed variability in future assets. Research is required to determine reusability metrics, and techniques are required to estimate and validate potential reuses. Hence, there is a mismatch between the procedures used in software project management and the demand for usable data from historical data. Data mining techniques can effectively and efficiently extract meaningful data using numerous criteria. The largest issue in software development is the development of reliable software and the determination of how to successfully extract modular elements that can be used to improve software quality. The methods now in use demonstrate the numerous attempts to identify reusable parts from software systems. There is an evolving need for a framework or mechanism that can precisely extract modules from huge and diverse large-scale datasets and enrich efficiency, especially with the rise of complex and massive O-O software systems.

A. *Objectives*

Software reusability has emerged as a potential tool for developing cost-effective and time-efficient software systems. This approach not only helps developers improve software quality but also addresses software complexity issues encountered during the software development process. The O-O approach not only provides a solution for the software industry to emerge from the crisis but also provides a dimension to enhance the quality of software systems. In the context of the increasing number of large and complex O-O software systems, there is a rising need for an approach or mechanism that facilitates the extraction of accurate reusable components from large datasets. This mechanism can contribute to improving the overall quality of software systems.

This study aimed to:

- Identify O-O metrics to study the reusability of software systems.
- Identify the most suitable data mining approaches to enhance the reusability of O-O software systems.
- Evaluate the performance of various data mining approaches based on precision, MSE, and accuracy rates.

B. *Self-Organizing Map (SOM)*

This algorithm is a well-known pattern recognition and classification technique. It belongs to the category of Artificial Neural Network (ANN) models and operates on the principle of competitive learning techniques. Additionally, it is classified as an unsupervised technique. SOM uses an organized array of neurons arranged in a two-dimensional grid. SOM maps the arbitrary dimensions of incoming input data signals onto a one- or two-dimensional space, enabling the exploration and comprehension of the inherent structure within the data.

II. RELATED WORK

Previous studies focused on various data mining approaches to improve the reusability of O-O software systems. In [2], a model was proposed to estimate the reusability of different software components, incorporating the Neuro-Fuzzy engine and the Chidamber & Kemerer (C&K) metrics. In [3], the H-SOM and Naïve Bayes approaches were compared using the R framework, showing that H-SOM was significantly more time-efficient compared to the Naïve Bayes approach. In [4], a novel approach was presented to quantify software reusability using a Support Vector Machine (SVM)-based classifier. The results showed the effectiveness of the SVM classifier in measuring software reusability, providing valuable insights for software developers looking to improve code reuse practices. In [5], clustering data mining was used to establish a software component repository. The algorithm primarily targeted functions, with a key emphasis on the specifications or behavior of these components. The specifications of the software components were represented through preconditions and postconditions, which were translated into first-order predicate logic before the clustering process. In [6], various methods and techniques used in data mining for classification tasks were explored, examining their effectiveness and applicability in diverse domains and offering valuable insights into the nuances of classification algorithms within the context of data mining. In [7], an innovative approach was introduced to extract software features from O-O source code, using an overlapping clustering technique. This method aimed to overcome the challenges associated with traditional feature extraction methods in O-O codebases. This approach identified and clustered software features considering their overlapping relationships, thereby providing a more comprehensive representation of the underlying software structure. Experimentation and analysis demonstrated the effectiveness of the overlapping clustering approach in accurately capturing software features, offering valuable insights for software engineering practitioners seeking to improve feature extraction processes in OO codebases. In [8], a method was proposed to identify reusable software modules from available software data, using genetic algorithms, the Rabin-Karp algorithm, and the K-means method, to facilitate the identification of software components that could be recycled, reducing costs, time, and effort. In [9], the concept of software reusability within software components was investigated to assess its practical implications and potential advantages. By examining a diverse array of software components, this study investigated the degree to which reusability is evident, the key factors that affect it, and its impact on software development processes. In [10], clustering methods were explored to restructure O-O software systems and improve their scalability and maintainability.

In [11], a novel approach for software cost estimation was presented, combining cuckoo optimization and K-Nearest Neighbors (KNN). This approach aimed to improve the accuracy and efficiency of software cost estimation by leveraging the complementary strengths of optimization and machine learning techniques. The performance of the hybrid algorithm was evaluated through empirical analysis and experimentation, providing valuable insights into its effectiveness compared to traditional methods. In [12], a novel approach was presented to improve the cohesion of various classes within O-O software systems by introducing a new similarity metric based on Frequent Usage Patterns (FUP). Using the values derived from this similarity metric, hierarchical agglomerative clustering was performed using a complete linkage strategy to group member functions into clusters. The primary objective of [13] was to introduce a software reuse approach and examine its impact on improving quality within the software industry.

In [14], various data mining rule-based classification techniques were applied to multiple datasets sourced from the NASA software repository, such as PC1 and PC2. The objective was to categorize software modules as either fault- or non-fault-prone. This study also modified the RIDOR algorithm, showing that it outperformed other classification techniques in terms of both the number of rules extracted and accuracy. In [15], a pool of 44 factors that could positively or negatively affect the reusability of software systems was refined. Regarding software metrics, this study presented findings across five primary categories of metrics: coupling, cohesion, complexity, inheritance, and size. The results showed that most of these metrics assessed reusability at the class level, with limited availability of software tools for analysis. Software reuse involves leveraging existing software artifacts to develop new systems, often with minimal or no modifications, to incorporate additional functionalities [16, 17]. In [16], a metric framework was used to perform a functional analysis of O-O-based software components, considering the maintainability index as a key factor in identifying crucial attributes to predict reusability levels. Unlike structural-level analysis, this method operated at the functional level, and its performance was evaluated based on precision, recall, accuracy, and classification error. Table I summarizes different data mining approaches used to enhance the reusability of O-O software systems.

TABLE I. COMPARATIVE ANALYSIS OF DIFFERENT DATA MINING TECHNIQUES

Ref.	Data MiningApproach	Input Dataset
[3]	HSOM, Naïve Bayes	Structured SE data
[4]	SVM	Software metrics value
[10]	Clustering approach (FCM)	Basis code of the software with classes
[12]	Hierarchical agglomerative clustering	JAVA projects
[18]	Decision Tree	O-O classes
[19]	Naïve Bayes and SVM	Software metrics data
[20]	Agglomerative clustering algorithm	Class members
[21]	Hierarchical Clustering (HC) and Non-HC	C++ modules
[22]	Neural Network	Software metrics data
[23]	SOM	CK metric values of Eclipse JDT, Eclipse PDE and Hibernate system

There is an increasing trend in the use of data mining approaches to enhance reusability. However, there are some significant research gaps, such as:

- Current strategies focus on reusing software artifacts based on expected functionality, but non-functional aspects are equally important.
- The major challenge in component-based software reuse is to efficiently manage a large cache of reusable elements to facilitate fast allocation and retrieval.
- Existing solutions are inadequate in addressing the various reusability levels of software components structured on O-O architecture.
- Existing reuse repository systems operate independently of manufacturing processes and tools, assuming software developers initiate the reuse process when required.
- Developers frequently needed to develop additional code for every project rather than relying on pre-existing, tested components, which poses a substantial hurdle to code reuse.

III. MATERIALS AND METHODS

This study aims to understand, evaluate, and project the impacts of software module metric values on the reusability of O-O software systems, incorporating the phases shown in Figure 1.

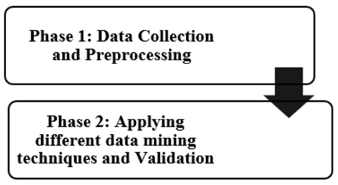
Phase 1: Data Collection and Preprocessing
Phase 2: Applying different data mining techniques and Validation

Fig. 1. Phases of the study.

A. Data Collection and Preprocessing

A dataset was collected, comprising 1000 instances from O-O software systems, namely Apache Tomcat, Eclipse, and Apache Sling, sourced from the MAVEN repository. These instances were prepared using the Data Extraction Algorithm shown in Algorithm 1.

ALGORITHM 1: DATA EXTRACTION

```
    //Load data into array G
1:  G = importdata("untitled4.mat")
    // For each relevant element in G that has a
    // missing value, produce an array containing
    // a logical 1
2:  Ix = ismissing(g, {'.' 'NA' -99})
    // Remove missing entries from G
3:  rowsWithMissing = g (any (ix,2))
4:  disp(rowsWithMissing)
    // Generate a standardized array by
    // substituting standard missing values in G
5:  G = standardizeMissing (g, -99)
6:  disp(G)
    // Fill missing values in G from corresponding
    // neighbor rows
7:  H= fillmissing(g, 'previous')
8:  disp(H)
```

The input dataset underwent further preprocessing to discern classes suitable for use as a unified form of instances. All records with a value of -1, representing missing data, were removed from the dataset [24]. Table II shows the number of classes before and after cleaning.

A semi-automated approach was employed to measure the software metrics values in the dataset. A dataset of O-O metrics and their corresponding reusability value for software components is required to evaluate the reusability of O-O software systems. This study incorporated O-O metrics as input to measure the relationship between metric values and reusability levels. This study used Message Passing Coupling (MPC), Response For a Class (RFC), Cohesion Among Methods of class (CAM), Lack of Cohesion (LCOM3), Weighted Methods per Class (WMC), and Data Access (DAC) metrics. The software components in all instances within the input dataset were evaluated using the software metrics. Figure 2 shows the distribution of the components in the three reusability classes: high, medium, and low. Analytical descriptions of these metrics are shown in Table III. The interpretation of analytical statistics in Table III is that the high mean value of DAC is associated with low productivity and the need for more efforts to make the available classes reusable. The significance of this finding is that it considers the strength of the main concepts relating to cohesion and coupling.

TABLE II. DATA PREPROCESSING

Software	Original classes	Classes with missing data	Cleaned classes
Apache Tomcat	285	85	200
Eclipse	210	60	150
Apache Sling	255	155	100

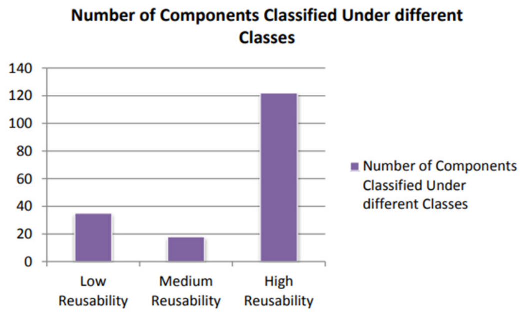

Fig. 2. Classification of components.

TABLE III. ANALYTICAL DESCRIPTION OF INPUT DATASET

	MPC	RFC	CAM	WMC	DAC
Minimum value	1	0	0	0	0
Maximum value	2066	28	17	240	8056
Standard deviation	173.67	5.91	4.89	25.56	2812.78
Mean value	72.56	0.78	5.67	27.67	315.67

B. *Applying Data Mining Techniques*

Various data mining techniques were applied to the input dataset after data preprocessing. Accuracy, Mean Squared Error (MSE), and precision were used to evaluate the performance of all data mining approaches.

IV. RESULTS AND DISCUSSION

The identification of highly reusable components is facilitated by data mining techniques with high precision, high recall, and low RSE rates. The precision of a class was calculated by dividing the total number of data elements belonging to the positive class by the number of valid positives, or the number of data elements accurately belonging to the positive class [25]. Figure 3 shows the precision values for the Bayesian classification, k-means, feature extraction, decision tree, backpropagation, agglomerative clustering, hierarchical clustering, and SOM approaches. When comparing the precision values, the SOM approach has a high precision of 37.8 % compared to the other techniques. The higher precision of the SOM technique contributes to better cohesion and reusability. The results also showed that the k-means technique may not be suitable for extracting the reusable components due to its low precision (35.5%). Extraction of appropriate reusable components will help improve the reusability of O-O software systems.

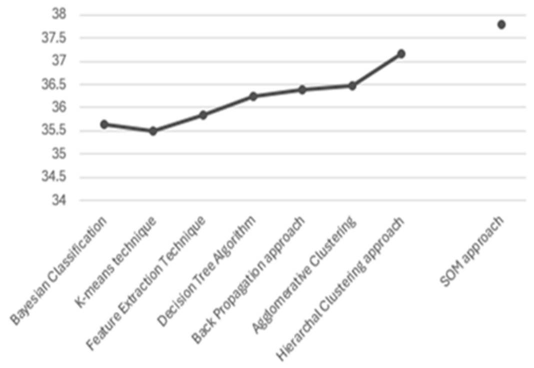

Fig. 3. The precision of various data mining techniques against precision.

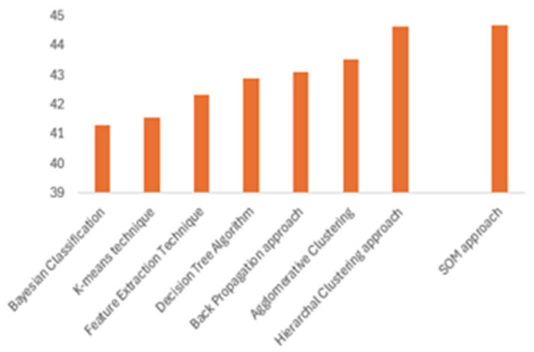

Fig. 4. Comparison of recall values for various data mining techniques.

Figure 4 compares the recall values for Bayesian classification, k-means, feature extraction, decision tree, backpropagation, agglomerative clustering, hierarchical clustering, and SOM. When comparing the recall values, SOM had a high recall of 44.8 %, compared to the other data mining techniques, contributing to better cohesion and reusability. High cohesiveness is preferred in O-O design, since it provides simplicity to comprehend, maintain, and reuse. The results also show that the Bayesian classification may not be suitable for extracting reusable components due to its low recall (41%).

RMSE is a commonly used metric for comparing the values [26] predicted by an estimator or model to those observed from the object being estimated or modeled. Figure 5 compares the RSME rates of Bayesian classification, k-means, feature extraction, decision tree, backpropagation, agglomerative clustering, hierarchical clustering, and SOM approaches. When comparing the RSME values, SOM had the lowest RSME of 0.4 % compared to the other data mining techniques. The low RMSE rate of the SOM technique results in the extraction of highly reusable modules. A low RMSE rate means that the extracted reusable components produce fewer errors.

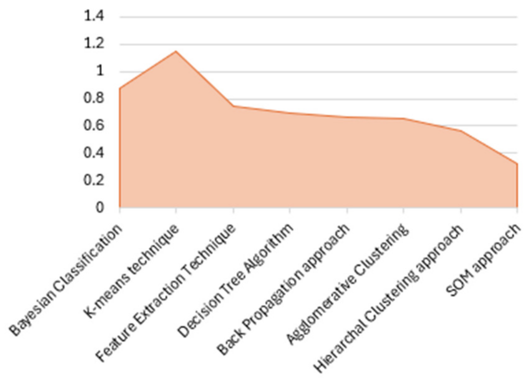

Fig. 5. RMSE comparison of various data mining techniques.

After extracting the input dataset from the MAVEN repository, a preprocessing step was implemented to identify missing values, check for noisy data, and address any other inconsistencies. Figure 6 shows the raw data containing missing values and outliers, which can potentially result in misleading conclusions during analysis.

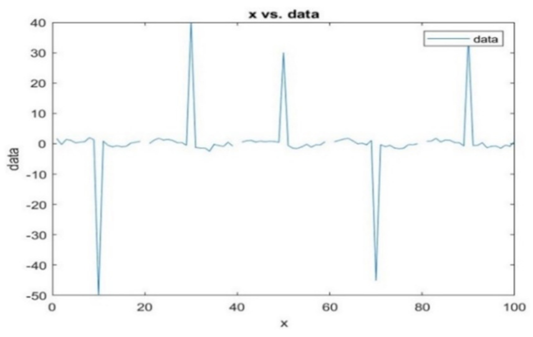

Fig. 6. Raw data representing missing values.

Although this study was able to perform well in analyzing various data mining approaches and find the most suitable one to improve reusability in O-O software components, it still has some limitations:

- To enhance the reliability of the results, the study can incorporate validation using multiple datasets with diverse metadata attributes.
- This study used only limited software metrics for the validation of data mining approaches. Including more software metrics and reuse metrics could improve accuracy.
- This study focused on the impact of data mining techniques on low and high reusability levels. However, it did not predict the effects of data mining techniques on medium reusability levels.
- This study lacks an effective model for identifying reusable software modules.

V. COMPARISON WITH PREVIOUS STUDIES

Table IV compares the proposed with the data mining approaches examined in [28] on the classification of low and high reusability instances, using accuracy, recall, and precision. Values are considered as depicted in [28].

TABLE IV. COMPARISON OF DATA MINING TECHNIQUES

Approaches	Results
K-means technique (Technique 1) [28]	Accuracy:67% Precision (Low Reusability Level): 0.8 (80%) Precision (High Reusability Level): 0.7(70%) Recall (Low Reusability Level): 0.91(91%) Recall (High Reusability Level): 0.9(90%)
DBSCAN (Technique 3) [28]	Accuracy:85% Precision (Low Reusability Level): NA Precision (High Reusability Level): 0.81(81%) Recall (Low Reusability Level): NA Recall (High Reusability Level): 0.93(93%)
JRA^{2}M^2 [28]	Accuracy:87.75 % Precision (Low Reusability Level): 0.88(88%) Precision (High Reusability Level): 0.96 (96%) Recall (Low Reusability Level): 0.8 (80%) Recall (High Reusability Level): 0.94 (94%)
SOM approach (This study)	Accuracy:91.2% Precision (Low Reusability Level): 0.9(90%) Precision (High Reusability Level): 0.98 (98%) Recall (Low Reusability Level): 0.86 (86%) Recall (High Reusability Level): 0.96 (96%)

VI. CONCLUSION AND FUTURE WORK

This study examined different data mining approaches used to improve the reusability of O-O software systems. Some key observations are as follows:

- Reusability is the premier strategy for improving software quality while reducing the expense and time needed for software development.
- O-O software components are far more reusable than component-based software systems.
- Classes are common components in O-O software that are easy to reuse.
- The results of the experimental analysis indicated that the SOM approach is a better data mining technique compared to the others.
- Pattern recognition and classification are well combined in the SOM algorithm [27].

Although software reusability helps to improve productivity and enhance software quality, it is still viewed as a challenging approach. This study examined several data mining strategies and tools, as well as a variety of recent studies on reusability prediction of O-O systems. The findings indicate that current solutions do not adequately address the diverse levels of reusability associated with O-O software components. Encouraging the reuse of components based on the expected level of reusability of newly developed or already available components can support software engineers and increase their productivity. This approach can offer time-to-market advantages for organizations and new products, since developers can enhance the efficacy, dependability, quality, and maintainability of their code. This approach could also result in significantly higher consumer satisfaction. Future research could focus on a hybrid approach that considers reusability variables such as design and code reuse. This research intends to add more features to the reusability enhancement framework for the assessment of other object-oriented languages, such as Python and C++.

ACKNOWLEDGMENT

The author expresses his gratitude to Parul Gandhi for providing continuous guidance during this work. This study did not receive a specific grant from any funding agency in the public, commercial, or non-profit sectors.

REFERENCES

[1] A. Alkhalid, M. Alshayeb, and S. A. Mahmoud, "Software refactoring at the class level using clustering techniques," *Journal of Research and Practice in Information Technology*, vol. 43, no. 4, pp. 285–306, Nov. 2011, https://doi.org/10.3316/ielapa.232430239489663.

[2] A. Shatnawi and A. D. Seriai, "Mining reusable software components from object-oriented source code of a set of similar software," in *2013 IEEE 14th International Conference on Information Reuse & Integration (IRI)*, San Francisco, CA, USA, Aug. 2013, pp. 193–200, https://doi.org/10.1109/IRI.2013.6642472.

[3] G. Maheswari and K. Chitra, "Enhancing Reusability and Measuring Performance Merits of Software Component using Data Mining," *International Journal of Innovative Technology and Exploring Engineering*, vol. 8, no. 6s4, pp. 1577–1583, Apr. 2019, https://doi.org/10.35940/ijitee.F1318.0486S419.

[4] A. Kumar, "Measuring Software reusability using SVM based classifier approach," *International Journal of Information Technology and Knowledge Management*, vol. 5, no. 1, pp. 205–209, 2012.

[5] S. Vodithala and S. Pabboju, "A clustering technique based on the specifications of software components," in *2015 International Conference on Advanced Computing and Communication Systems*, Coimbatore, India, 2015, pp. 1–6, https://doi.org/10.1109/ICACCS. 2015.7324095.

[6] K. Deeba and B. Amutha, "Classification Algorithms of Data Mining," *Indian Journal of Science and Technology*, vol. 9, no. 39, Oct. 2016, https://doi.org/10.17485/ijst/2016/v9i39/102065.

[7] I. E. Araar and H. Seridi, "Software Features Extraction From Object-Oriented Source Code Using an Overlapping Clustering Approach," *Informatica*, vol. 40, no. 2, pp. 245–255, 2016.

[8] C. Gupta and M. Rathi, "A Meta Level Data Mining Approach to Predict Software Reusability," *International Journal of Information Engineering and Electronic Business*, vol. 5, no. 6, pp. 33–39, Dec. 2013, https://doi.org/10.5815/ijieeb.2013.06.04.

[9] M. Arifa, N. Mohamed, and K. Archana, "Study of Software Reusability in Software Components," *International Journal of Information and Communication Technology,* vol. 5, no. 7, pp. 2455-2460, Aug. 2013.

[10] S. Bobde and R. Phalnikar, "Restructuring of Object-Oriented Software System Using Clustering Techniques," in *Proceeding of International Conference on Computational Science and Applications*, Saint Petersburg, Russia, Jul. 2019, pp. 419–425, https://doi.org/10.1007/978-981-15-0790-8_41.

[11] E. E. Miandoab and F. S. Gharehchopogh, "A Novel Hybrid Algorithm for Software Cost Estimation Based on Cuckoo Optimization and K-Nearest Neighbors Algorithms," *Engineering, Technology & Applied Science Research*, vol. 6, no. 3, pp. 1018–1022, Jun. 2016, https://doi.org/10.48084/etasr.701.

[12] A. Rathee and J. K. Chhabra, "Restructuring of Object-Oriented Software Through Cohesion Improvement Using Frequent Usage Patterns," *ACM SIGSOFT Software Engineering Notes*, vol. 42, no. 3, Sep. 2017, https://doi.org/10.1145/3127360.3127370.

[13] A. Mateen, S. Kausar, and A. R. Sattar, "A Software Reuse Approach and Its Effect On Software Quality, An Empirical Study for The Software Industry," *International Journal of Management, IT & Engineering*, vol. 7, no. 2, pp. 266–279, Feb. 2017, https://doi.org/10.48550/arXiv. 1702.00125.

[14] N. Hassan and I. Alsmadi, "Enhance Rule Based Detection for Software Fault Prone Modules," *International Journal of Software Engineering and Its Applications,* vol. 6, no. 1, pp. 75-86, Jan., 2012.

[15] B. Mehboob, C. Chong, S. Lee, and J. Lim, " Reusability affecting factors and software metrics for reusability: A systematic literature review," *Journal of Software: Practice and Experience,* vol. 51, no. 6, pp. 1416-1458, Mar. 2021.

[16] C. Gupta, S. Maggo, "An efficient prediction model for software reusability for Java-based object-oriented systems," *International Journal of Computer Aided Engineering and Technology*, vol. 6, no. 2, pp. 182-199, Jan. 2014, https://doi.org/10.1504/IJCAET.2014.060299.

[17] B. Bisht and P. Gandhi, "Software Reusability of Object-Oriented Systems using Data Mining Techniques," *International Journal of Recent Technology and Engineering (IJRTE)*, vol. 8, no. 6, pp. 2144–2152, Mar. 2020, https://doi.org/10.35940/ijrte.F8155.038620.

[18] P. Divanshi and A. Singh, "A Classification-based Predictive Cost Model for Measuring Reusability Level of Open Source Software," *International Journal of Recent Research Aspects*, vol. 5, no. 1, pp. 19-23, Jan. 2018.

[19] M. Gupta and S. Singh, "Empirical Evaluation of Software Design Patterns using Classification Algorithms based Design Metrics," *International Journal of Applied Engineering Research*, vol. 13, no. 15, pp. 11816–11823, 2018.

[20] M. Fokaefs, N. Tsantalis, A. Chatzigeorgiou, and J. Sander, "Decomposing object-oriented class modules using an agglomerative clustering technique," in *2009 IEEE International Conference on Software Maintenance*, Edmonton, Canada, Sep. 2009, pp. 93–101, https://doi.org/10.1109/ICSM.2009.5306332.

[21] J. Bhagwan and A. Oberoi, "Software Modules Clustering: An Effective Approach for Reusability," *Journal of Information Engineering and Applications*, vol. 1, no. 4, 2011.

[22] S. Manhas, R. Vashisht, P. S. Sandhu, and N. Neeru, "Reusability Evaluation Model for Procedure Based Software Systems," *International Journal of Computer and Electrical Engineering*, pp. 1107–1111, 2010, https://doi.org/10.7763/IJCEE.2010.V2.283.

[23] P. N. Smyrlis, D. C. Tsouros, and M. G. Tsipouras, "Constrained K-Means Classification," *Engineering, Technology & Applied Science Research*, vol. 8, no. 4, pp. 3203–3208, Aug. 2018, https://doi.org/10.48084/etasr.2149.

[24] W. B. Frakes and K. Kang, "Software reuse research: status and future," *IEEE Transactions on Software Engineering*, vol. 31, no. 7, pp. 529–536, Jul. 2005, https://doi.org/10.1109/TSE.2005.85.

[25] N. S. Gill, "Importance of software component characterization for better software reusability," *ACM SIGSOFT Software Engineering Notes*, vol. 31, no. 1, pp. 1–3, Jan. 2006, https://doi.org/10.1145/ 1108768.1108771.

[26] R. Keswani, S. Joshi, and A. Jatain, "Software Reuse in Practice," in *2014 Fourth International Conference on Advanced Computing & Communication Technologies*, Rohtak, India, Feb. 2014, pp. 159–162, https://doi.org/10.1109/ACCT.2014.57.

[27] V. R. Basili, L. C. Briand, and W. L. Melo, "How reuse influences productivity in object-oriented systems," *Communications of the ACM*, vol. 39, no. 10, pp. 104–116, Oct. 1996, https://doi.org/10.1145/236156. 236184.

[28] S. Maggo and C. Gupta, "MLP based Reusability Assessment Automation Model for Java based Software Systems," *International Journal of Modern Education and Computer Science*, vol. 6, no. 8, pp. 45–58, Aug. 2014, https://doi.org/10.5815/ijmecs.2014.08.06.