Int. j. inf. tecnol. (June 2022) 14(4):2013–2019
https://doi.org/10.1007/s41870-021-00846-0

ORIGINAL RESEARCH

Ensemble clustering based approach for software architecture recovery

Shiva Prasad Reddy Puchala[1] · Jitender Kumar Chhabra[1] · Amit Rathee[2]

Received: 1 August 2021 / Accepted: 2 December 2021 / Published online: 25 January 2022

Abstract Frequent software maintenance usually results in a change in the original software architecture and thus architecture recovery is attempted using dependencies in the software artifacts. Many different techniques for architecture recovery are proposed in the literature normally using one out of structural, semantic, and directory dependencies, and applying clustering over these computed values. The main drawback of these approaches is that either they use only one type of dependency computed in a limited way, and or merge multiple dependencies before applying a clustering technique and thus the approaches result in loss of information because each dependency has its unique characteristics. This paper proposes a new approach for architecture recovery using ensemble clustering and utilizing a more precise computation of three types of dependencies: structural, semantic, and directory dependencies. The proposed approach is evaluated over open-source java projects and analyzed. The obtained results clearly show that our proposed architecture recovery approach using ensemble clustering and multiple dependencies performs much better than other conventional recovery approaches.

Keywords Architecture recovery · Directory dependencies · Multiple dependencies · Ensemble clustering

✉ Shiva Prasad Reddy Puchala
puchalashivaprasadreddy@gmail.com

[1] Computer Engineering Department, National Institute of Technology, Kurukshetra 136119, India

[2] Computer Science Department, Government College, Barota, Gohana, Sonipat 131301, India

1 Introduction

Softwares are initially developed with a good architecture but subsequent changes in the code disturb the architecture and usually, the corresponding documentation remains un-updated. Proper software architecture is crucial for obvious reasons such as reuse, maintenance, etc. Hence, many researchers target architecture recovery by extracting different dependencies from the source code and applying clustering techniques to obtain a module view of software. Based on the dependencies used in architecture recovery these techniques are classified as, structure-based techniques, semantic-based techniques, and knowledge-based techniques [1]. Structure-based recovery techniques obtain structural dependencies by analyzing the structure of the source code and finding relations between entities. Semantic-based recovery techniques utilize the textual information like comments, documentation, and naming available in the source code, and apply information retrieval techniques to group entities. Knowledge-based recovery techniques obtain various dependencies from other sources such as change-history, directory structure, etc.

Most of the prevalent recovery techniques utilize the dependencies by combining them before applying clustering techniques and this results in poor usage of dependencies because every dependency has its unique characteristic. To overcome the restriction of using a single clustering technique and to be able to use multiple dependencies and clustering techniques effectively this paper proposes a new approach for software architecture recovery using ensemble clustering by utilizing multiple dependencies in the recovery process. The main contributions of this paper include:

1. Effective extraction of structural, semantic, and directory dependencies from the software source code.
2. A new efficient approach for software architecture recovery using ensemble clustering and multiple dependencies.
3. Evaluating the effect of individual dependencies and multiple dependencies in architecture recovery using ensemble clustering.

2 Related works

The software architecture recovery techniques can be discussed based on different types of dependencies extracted from the source code and then used for the recovery.

2.1 Architecture recovery using structural dependencies

Initially, architecture recovery was attempted using structural dependencies-based clustering through the Bunch tool [2]. Then Maqbool and Babri [3] proposed a weighted combined algorithm (WCA), a hierarchical clustering algorithm for architecture recovery based on the inter-cluster distance. Another hierarchical clustering algorithm was proposed as the scalable information bottleneck algorithm (LIMBO) [4] using information loss measures. Then the concept of fuzzy sets was extended to it in the form of a fuzzy logic-based hierarchical clustering technique named LBFHC [5]. Subsequently Zhang et al. [6] integrated partitioned and hierarchical clustering techniques and proposed a hybrid architecture recovery technique that uses class graphs to find the kernels and then partitions the graph based on the kernels. Recently Cho et al. [7] proposed an architecture recovery approach using cluster ensembles, which uses the results of different architecture recovery approaches and consolidates these results to recover a better architecture. All of these approaches used only a few types of structural dependencies, although 8 different types of structural dependencies exist among various software entities [8, 9] and all should be used for more accurate computation.

2.2 Architecture recovery using semantic dependencies

In literature, researchers leveraged the semantic information found in the software repositories and proposed architecture recovery techniques that could take advantage of the available linguistic information. Kuhn et al. [10] proposed an architecture recovery approach using information retrieval techniques, using Latent semantic indexing (LSI). A concern is defined as a concept or role of a software system and semantic dependencies are used in extracting them. Garcia et al. [11] proposed an architecture recovery technique using concerns by identifying connectors and components in a software system. Sajnani [12] proposed an automated architecture recovery approach using unsupervised learning methods. Link et al. [13] extended the concept of software concerns for architecture recovery approach by extracting software concerns from the textual information of source code and applied classification techniques to relate entities to a domain. Sun et al. [14] proposed a novel program comprehension approach based on Latent Dirichlet Allocation (LDA) model for clustering large-sized packages. Lee et al. [15] proposed an approach to improve the semantic dependencies-based architecture recovery by identifying and removing the semantic outliers from source code artifacts. Limited works have been done for recovering the architecture in this direction, but it is logical to say that semantic information is necessary, but not sufficient for the architecture recovery.

Observing the non-sufficiency of structural and semantic dependencies, some attempts have been recently made in the literature for architecture recovery techniques by using alternate information from software repositories such as knowledge, change history, directory structure, etc. Li et al. [16] proposed a framework information-oriented architecture recovery technique by considering framework-related patterns and features in clustering. Shahbazian et al. [17] proposed an approach to recover architectural design decisions using the code history found in version control systems. Kong et al. [1] proposed an architecture recovery approach based on the directory structure of software and generated a code dependency graph to apply traditional recovery techniques to the graph. Most recent being language-independent pattern-oriented architecture recovery framework by Guimaraes [18], which extracted design patterns and decisions from the source code for the architecture recovery. These alternate approaches did not give proper importance to all aspects of structural and semantic information and hence have limited applicability.

As evident from above, most of the research till date is confined to use single clustering based on one or at most two types of dependencies, and those also computed without covering all of their relevant types. Zahid et al. [19] surveyed evolution in software architecture recovery techniques and showed that existing techniques mainly focus on the extraction of components and connectors involved in the architecture by forming clusters and minimal attention was paid to concerns, architectural style, or design pattern used to develop the software system. Further, attempts of combining dependencies were done for a maximum of two types, and combining was done before clustering resulting in loss of unique information of each type. So all existing approaches are limited to using a single clustering technique and have the risk of the information loss while merging different dependencies. These research gaps and limitations of previous approaches motivated us to propose a new approach for software architecture recovery using ensemble clustering and multiple dependencies, computed in a precise manner from source code as well as other relevant structures.

3 Proposed approach

This section provides a detailed overview of the proposed architecture recovery approach using cluster ensembles and multiple dependencies.

Figure 1 depicts the procedure adopted in our proposed approach for software architecture recovery. First of all structural, semantic, and directory dependencies are extracted between software elements by analyzing the source code. Then different clustering techniques are applied using dependencies obtained above to generate clusterings. The intermediate clusterings are named base clusterings. Now base clusterings are consolidated using a consensus technique to obtain a similarity matrix and the complete linkage clustering is applied to this matrix to get the final clusterings. Finally, the obtained clustering results are evaluated with Turbo MQ and MoJoFM metrics.

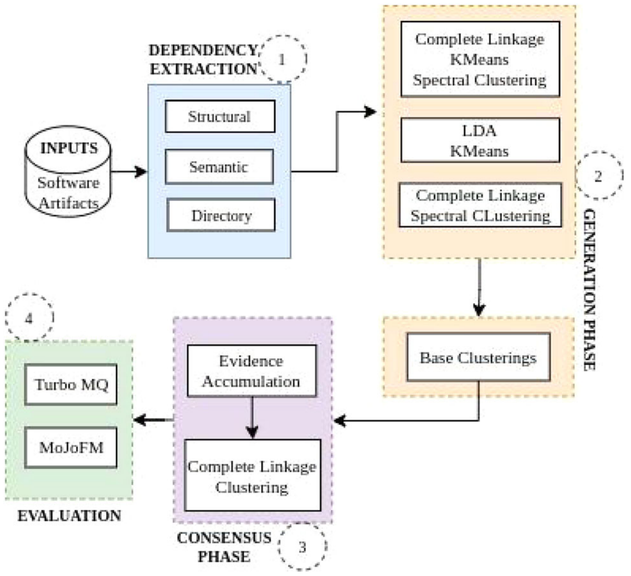

Fig. 1 The proposed ensemble clustering based approach for software architecture recovery

3.1 Improved and effective extraction of structural and semantic dependencies

In this paper, we considered package names, usage patterns, identifiers, class/method/variable definitions, statements, and comments to effectively mine the semantic relations. To extract the structural and semantic relations from the source code, the proposed mathematical approach by Rathee and Chhabra [9] is used in this paper and dependency matrices are generated for structural and semantic dependencies. The existing directory dependency processing approaches do not consider the file hierarchy found in the directory structure of software, so we improve this approach by considering the depth of each file in the file hierarchy and using it in the computation of directory dependencies. The process involved in the extraction of directory dependencies is described in the next section.

3.2 Extracting directory dependencies

Kong et al. [1] described that the software architecture belongs to the logical view and most of the approaches in literature ignore the available knowledge of directory structure while recovering the architecture. This paper proposes to use the directory structure also. The path information of each class file in the software system is extracted and this information is used in building a directory tree. The internal nodes in the directory tree represent intermediate directories and leaf nodes represent the classes in the system. The directory dependency between two classes is computed based on their placement in the directory tree. Generally developers tend to create and organize files in hierarchies while designing. So files at greater depth in the hierarchy are more related than the files at a lower depth. To reduce the effect of trivial distance-based measures and account for granularity, we consider depth in the similarity computation and it is calculated as the distance between the root node and common ancestor node [20]. The distance-depth based measure for similarity computation is expressed as:

$$S_D(C_i, C_j) = \frac{1 + D(A_{i,j}, R)}{1 + D(A_{i,j}, R) + D(C_i, C_j)} \tag{1}$$

Here, $D(C_i, C_j)$ is the length of path/distance between tree nodes of classes C_i and C_j, $A_{i,j}$ is the lowest common ancestor of (C_i, C_j) and R is the root. By using Eq. (1) the directory dependency matrix (DrDM) is calculated as:

$$DrDM(i,j) = \sum_{i=1}^{|N|} \sum_{j=1}^{|N|} S_D(C_i, C_j) \tag{2}$$

Here, |N| is the total number of classes in the system. DrDM is a symmetric matrix and its values vary between 0 and 1.

3.3 Clustering

Clustering techniques group entities based on their similarities to form clusters or sub-clusters. A clustering ensemble aims at combining multiple clustering results to produce a better result than that of individual clustering techniques and its effectiveness depends on the quality and quantity of intermediate clusterings or base clusterings obtained and used. There are two techniques to generate diverse base clusterings; (1) Using the same individual clustering technique with distinct parameter settings and (2) Adopting the suitable clustering technique for each base clusterings generation.

Most of the recovery techniques in literature which use structural dependencies either adopt Complete Linkage HAC [21] or KMeans as their clustering technique, so in this work, we used Complete Linkage HAC, KMeans, Spectral Clustering on structural dependencies to generate base clusterings. Garcia et al. [11] worked on semantic information-based architecture recovery using latent Dirichlet allocation (LDA) and it is also reported in the literature that LDA is modeled for finding topics in textual information. So we use LDA and KMeans on our semantic dependencies to generate base clusterings. On directory dependencies, we apply similarity-based clustering techniques such as Complete Linkage HAC and Spectral clustering to generate base clusterings.

Cho et al. [7] showed that for better usage of cluster ensemble approach in software architecture recovery the number of base clustering generated should be above 20 and normally between 20 and 70 to avoid oversaturation. Accordingly, we chose to generate a total of 31 base clusterings; as shown in Table 1 for our experiments.

Table 1 Parameter setting for base clusterings generation

Dependency	Clustering technique	# of base clusterings
Structural	Complete linkage HAC	5
	Kmeans	5
	Spectral clustering	5
Semantic	Latent Dirichlet allocation	5
	KMeans	5
Directory	Complete linkage HAC	3
	Spectral clustering	3
Total # of base clusterings		31

3.4 Consolidation of base clusterings using consensus techniques

In this step, consensus techniques are applied to the generated base clusterings to consolidate them into a unique clustering result. As described by Cho et al. [7] there are several consensus techniques in literature; we adopt the evidence accumulation (EA) algorithm [22] as our consensus technique. Evidence accumulation algorithm is shown to perform well in the literature and it works on the entity co-occurrence approach. It uses the number of co-occurrences across base clusterings as similarities between two entities and generates a similarity matrix. Finally complete linkage HAC is applied to the similarity matrix to produce the final clustering results.

4 Experimentation and results

We experimented with our approach on two open-source projects and analyzed the results of applying our approach by using individual dependencies and multiple dependencies. Finally, we compared the evaluation scores of applying our approach to the weight-based dependency clustering approach.

4.1 Formulated research questions

In this paper we answer the following two research questions:

RQ1 Do multiple dependencies perform better than individual dependencies in recovering architecture using ensemble clustering?

RQ2 Do software architecture recovery using ensemble clustering and multiple dependencies perform better than approaches using weight-based dependency clustering?

4.2 Subject projects

Two open-source projects ArchStudio and Hadoop are selected to investigate the two research questions. Lutellier et al. [23] constructed expert architectures of five projects in their work. The projects are Chromium, ITK, Bash, Hadoop, ArchStudio and they provided open access to the expert decompositions of these projects. So we selected Hadoop and ArchStudio as our subject systems because of their accessible expert decompositions. The clustering techniques require the parameter i.e. the number of clusters to be set in advance, so in this paper, we decided the number of clusters parameter by examining the clusters in expert decompositions.

4.3 Evaluation measures

In this paper, the clustering results obtained from the consensus phase are evaluated with MoJoFM [24] and Turbo MQ [25] measures. MoJoFM and Turbo MQ have been commonly used in the literature for the comparison of architectures and clusters. These metrics are shortly summarized as follows:

(i) *MoJoFM:* MoJoFM is an external evaluation measure. It measures the similarity of recovered architecture to the expert architecture and it is calculated as,

$$MoJoFM = \left(1 - \frac{mno(P,Q)}{max(mno(\forall P,Q))}\right) \times 100\% \quad (3)$$

P is the recovered architecture, Q is expert architecture, mno(P, Q) is the minimum number of Move and Join operations needed to transform P to Q. A lower score indicates a greater disparity between the architectures P and Q, a higher score indicates how much P is closer to Q.

(ii) *Turbo modularization quality (turbo MQ):* Turbo MQ is an internal quality measure that is based on the assumption that recovered architecture should exhibit high cohesion and low coupling. Turbo MQ is calculated as,

$$TurboMQ = \sum_{i=1}^{k} \frac{2\mu_i}{2\mu_i + \sum_j (\varepsilon_{ij} + \varepsilon_{ji})} \quad (4)$$

where μ_i indicates the number of intra-relationships in cluster i, and $\varepsilon_{ij} + \varepsilon_{ji}$ indicates the number of inter-relationships between cluster i and cluster j.

4.4 Results and analysis

The section presents the results obtained after evaluating the proposed approach on ArchStudio and Hadoop software systems. In our experimentations, we applied complete linkage clustering on the similarity matrix obtained after the consensus phase to generate the final architectural decompositions. Finally, the recovered decompositions are evaluated with Turbo MQ and MoJoFM evaluation measures, and the scores are presented in Table 2. For ArchStudio, we experimented exhaustively with 10 different numbers of clusters as 53, 54,…,62, and for Hadoops, 62, 63,…,71. We have also presented the evaluation scores of using individual dependencies in the ensemble clustering based architecture recovery approach in Table 2. The following table shows some useful results for a selective number of clusters.

RQ1 Do multiple dependencies perform better than individual dependencies in recovering architecture using ensemble clustering?

Table 2 Few results of experimentation

#[a]	STR[b]	SEM[b]	DIR[b]	Our App[b]
ArchStudio				
53	39.2, 7.4	53.6, 12.7	80.1, 19.0	76.2, 20.4
57	42.27, 8.0	54.53, 13.6	80.28, 19.5	75.67, 21.1
62	47.78, 10.9	56.13, 14.5	81.53, 21.1	75.31, 21.7
Hadoop				
62	30.44, 13.5	37.88, 11.0	62.12, 15.3	52.92, 19.1
69	30.97, 15.0	40.35, 12.4	63.01, 15.2	53.45, 19.8
71	30.62, 15.2	40.35, 12.5	62.83, 15.3	53.1, 19.8

[a]# of clusters

[b]Values of MojoFM, TurboMQ

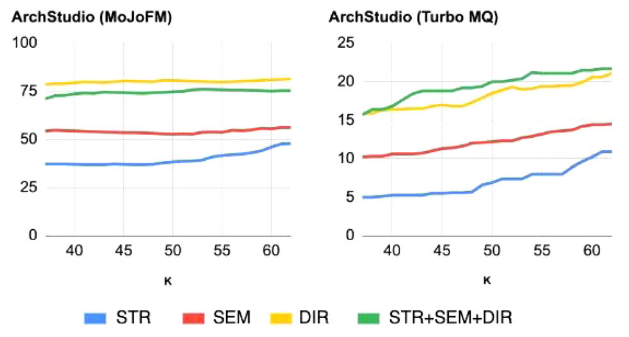

Fig. 2 Comparison of the evaluation scores obtained using individual dependencies and multiple dependencies using ensemble clustering approach

Figure 2 shows a comparison of results obtained using each dependency individually and multiple dependencies in architecture recovery using ensemble clustering approach, From the plots in Fig. 2, it is observed that the ensemble clustering approach using directory dependencies shows higher scores of MoJoFM than any other dependencies because for the selected subject system the existing directory structure is well maintained, but it can be observed that the scores of Turbo MQ using directory dependencies are lower than others because the directory dependencies alone cannot account for the cohesiveness of the entities present in a cluster. The obtained scores of MoJoFM and Turbo MQ by applying ensemble clustering approach on Archstudio and Hadoop show that the results of a recovery technique improve by using multiple dependencies and it is also clear that the cohesiveness of clusters formed by using multiple dependencies is much better compared to others. As we can see from Fig. 2, the Turbo MQ scores of each subject system by using structural, semantic, and directory dependencies together in architecture recovery would eventually improve and outperform other individual dependencies.

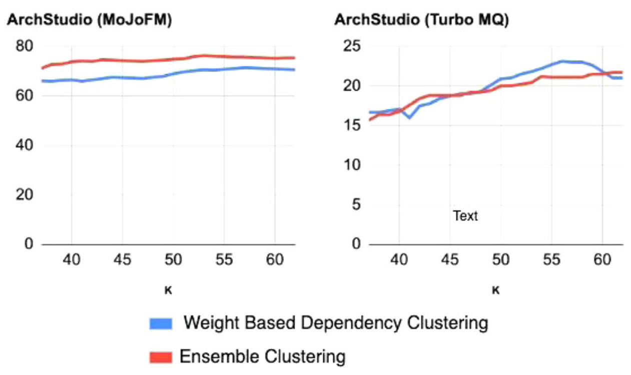

Fig. 3 Comparison of the evaluation scores obtained using weight-based dependency clustering and ensemble clustering approach

RQ2 Do software architecture recovery using ensemble clustering and multiple dependencies perform better than approaches using weight-based dependency clustering?

To test RQ2, we obtained the software dependency matrices of ArchStudio for each type of dependency and combined them using the following 0.5, 0.3, 0.2 weights for structural, semantic, and directory dependencies respectively. Then we applied complete linkage HAC to get the final clustering results and evaluated results with MoJoFM and Turbo MQ. The same approach is also repeated for the subject system Hadoop and the results are noted. Finally, the scores obtained using weight-based dependency scheme and ensemble clustering scheme on the subject system ArchStudio are plotted in Fig. 3. The proposed approach eliminates the burden of setting the weights for each type of dependency manually as in the weight-based dependency clustering.

From Fig. 3 it can be observed that for subject system ArchStudio, the proposed approach continued to perform well in the case of both MoJoFM and Turbo MQ. Finally, based on observations from the subject system Hadoop and the plots of ArchStudio shown in Fig. 3, it is concluded that architecture recovery using ensemble clustering and multiple dependencies outperforms weight-based dependency clustering schemes in architecture recovery.

4.5 Comparison with existing approaches in the literature

To show how an ensemble-based clustering approach improves the architecture recovery results, a comparison is provided between Kong et al. [1], Cho et al. [7] and the proposed approach. Cho et al. [7] proposed an architecture recovery approach based on cluster ensembles, and Kong et al. [1] proposed an approach that generates a submodule-level dependency graph based on directory hierarchy and uses it to improve the architecture recovery. Table 3 presents the scores of MoJoFM and Turbo MQ obtained after evaluating the proposed approach and the values presented by Cho et al. [7] and Kong et al. [1] approaches on Hadoop and ArchStudio. The main reason behind considering ArchStudio and Hadoop is that these software systems are also evaluated by Cho et al. [7] and Kong et al. [1] and their proposed approaches are not publicly available for experimentation purposes.

In the case of ArchStudio, it can be observed from Table 3 that even with a lower number of base clusterings the proposed approach showed an improvement of 8–13% in the MoJoFM evaluation scores. This is because the proposed approach effectively extracts dependencies and accurately uses them in the recovery process. In the case of Hadoop, we can observe a minor improvement in the scores of MojoFM and Turbo MQ and our observations show that the cohesiveness of the clusters formed by our approach is better. The recovery approach by Cho et al. [7] is not directly available for evaluation, so deep further analysis is needed and being carried out. Finally, based on the results of subject systems Hadoop and ArchStudio, it can be concluded that the proposed approach performs better than Cho et al. [7] and Kong et al. [1] approaches and improves the overall architecture recovery results.

5 Conclusion

Program comprehension and software maintenance can be done in a much better way using the software architecture derived from the most recent implementation than the older versions. Various dependencies and their integration play an important role in this process. In this paper, a new approach to architecture recovery is proposed using ensemble clustering and multiple dependencies. The

Table 3 Comparison of the proposed approach with Kong et al. [1] and Cho et al. [7] approaches

Arch. recov. approaches	Hadoop		ArchStudio	
	MoJoFM (%)	Turbo MQ	MoJoFM (%)	TurboMQ
Kong et al.'s [1]	49	18.1	62	26.35
Cho et al. [7]	53	–	67	–
Proposed approach	53	19.8	75	21.7

structural, semantic, and directory dependencies are effectively utilized in the architecture recovery process and each dependency is fully leveraged by applying suitable clustering techniques on it. The proposed approach also eliminates the restriction of relying on the results of a single clustering technique and the application of different clustering techniques on each dependency helps in generating diverse base clusterings which in turn improves the accuracy of architecture recovery. Our experimentations verified that architecture recovery using ensemble clustering and multiple dependencies performs much better than weight-based dependency clustering.

In the future, the proposed approach can be investigated by considering different dependencies and clustering techniques by generating more diverse base clusterings to improve the recovery results. A more detailed evaluation can be carried out to study the architecture recovery results by applying various consensus techniques. The proposed ensemble clustering based approach for architecture recovery can be utilized for software remodularization and in software restructuring.

References

1. Kong X, Li B, Wang L, Wu W (2018) Directory-based dependency processing for software architecture recovery. IEEE Access 6:52321–52335. https://doi.org/10.1109/access.2018.2870118
2. Mancoridis S, Mitchell B, Chen Y, Gansner E (1999) Bunch: a clustering tool for the recovery and maintenance of software system structures. In: Proceedings IEEE international conference on software maintenance—1999 (ICSM'99). 'software maintenance for business change' (Cat. No.99CB36360), pp 50–59. https://doi.org/10.1109/icsm.1999.792498
3. Maqbool O, Babri H (2004) The weighted combined algorithm: a linkage algorithm for software clustering. In: 8th European conference on software maintenance and reengineering, 2004. CSMR 2004. Proceedings, pp 15–24. https://doi.org/10.1109/csmr.2004.1281402
4. Andritsos P, Tzerpos V (2005) Information-theoretic software clustering. IEEE Trans Software Eng 31(2):150–165. https://doi.org/10.1109/tse.2005.25
5. Wang Y, Liu P, Guo H, Li H, Chen X (2010) Improved hierarchical clustering algorithm for software architecture recovery. In: 2010 international conference on intelligent computing and cognitive informatics, pp 247–250.https://doi.org/10.1109/icicci.2010.45
6. Zhang Q, Qiu D, Tian Q, Sun L (2010) Object-oriented software architecture recovery using a new hybrid clustering algorithm. In: 7th international conference on fuzzy systems and knowledge discovery, vol 6, pp 2546–2550. https://doi.org/10.1109/fskd.2010.5569799
7. Cho C, Lee K, Lee M, Lee C (2019) Software architecture module-view recovery using cluster ensembles. IEEE Access 7:72872–72884. https://doi.org/10.1109/access.2019.2920427
8. Prajapati A, Parashar A, Chhabra J (2020) Restructuring object-oriented software systems using various aspects of class information. Arab J Sci Eng 45(12):10433–10457. https://doi.org/10.1007/s13369-020-04785-z
9. Rathee A, Chhabra J (2018) Clustering for software remodularization by using structural, conceptual and evolutionary. J JUCS 24(12):1731–1757. https://doi.org/10.3217/jucs-024-12-1731
10. Kuhn A, Ducasse S, Gîrba T (2007) Semantic clustering: identifying topics in source code. Inf Softw Technol 49(3):230–243. https://doi.org/10.1016/j.infsof.2006.10.017
11. Garcia J, Popescu D, Mattmann C, Medvidovic N, Yuanfang C (2011) Enhancing architectural recovery using concerns. In: 26th IEEE/ACM international conference on automated software engineering (ASE 2011), pp 552–555. https://doi.org/10.1109/ase.2011.6100123
12. Sajnani H (2012) Automatic software architecture recovery: a machine learning approach. In: 20th IEEE international conference on program comprehension (ICPC), pp 265–268. https://doi.org/10.1109/icpc.2012.6240501
13. Link D, Behnamghader P, Moazeni R, Boehm B (2019) Recover and RELAX: concern-oriented software architecture recovery for systems development and maintenance. In: 2019 IEEE/ACM international conference on software and system processes (ICSSP), pp 64–73.https://doi.org/10.1109/icssp.2019.00018
14. Sun X, Liu X, Li B, Li B, Lo D, Liao L (2017) Clustering classes in packages for program comprehension. Sci Program 2017:1–15. https://doi.org/10.1155/2017/3787053
15. Lee K, Lee C (2020) Identifying semantic outliers of source code artifacts and their application to software architecture recovery. IEEE Access 8:212467–212477. https://doi.org/10.1109/access.2020.3040024
16. Li X, Zhang L, Ge N (2017) Framework information based java software architecture recovery. In: 24th Asia-Pacific software engineering conference workshops (APSECW), pp 114–120. https://doi.org/10.1109/apsecw.2017.15
17. Shahbazian A, Kyu Lee Y, Le D, Brun Y, Medvidovic N (2018) Recovering architectural design decisions. In: 2018 IEEE international conference on software architecture (ICSA), pp 95–9509.https://doi.org/10.1109/icsa.2018.00019
18. Guimaraes E, Cai Y (2020) Understanding software systems through interactive pattern detection. In: 2020 IEEE international conference on software architecture companion (ICSA-C), pp 51–54. https://doi.org/10.1109/icsa-c50368.2020.00017
19. Zahid M, Mehmmod Z, Inayat I (2017) Evolution in software architecture recovery techniques—a survey. In: 13th international conference on emerging technologies (ICET), pp 1–6. https://doi.org/10.1109/icet.2017.8281704
20. Sologub (2011) On measuring of similarity between tree nodes. In: Proceedings of the 5th Russian young scientists conference in information retrieval
21. Maqbool O, Babri H (2007) Hierarchical clustering for software architecture recovery. IEEE Trans Software Eng 33(11):759–780. https://doi.org/10.1109/tse.2007.70732
22. Fred A, Jain A (2005) Combining multiple clusterings using evidence accumulation. IEEE Trans Pattern Anal Mach Intell 27(6):835–850. https://doi.org/10.1109/tpami.2005.113
23. Lutellier T, Chollak D, Garcia J, Tan L, Rayside D, Medvidovic N, Kroeger R (2015) Comparing software architecture recovery techniques using accurate dependencies. In: 2015 IEEE/ACM 37th IEEE international conference on software engineering, pp 69–78. https://doi.org/10.1109/icse.2015.136
24. Zhihua W, Tzerpos V (2004) An effectiveness measure for software clustering algorithms. In: 12th IEEE international workshop on program comprehension proceedings, pp 194–203. https://doi.org/10.1109/wpc.2004.1311061
25. Mitchell B (2003) A heuristic approach to solving the software clustering problem. In: International conference on software maintenance, 2003. ICSM 2003. Proceedings, pp 285–288.https://doi.org/10.1109/icsm.2003.1235432