International Journal of Information System Modeling and Design
Volume 13 • Issue 1

Software Architecture Recovery Using Integrated Dependencies Based on Structural, Semantic, and Directory Information

Shiva Prasad Reddy Puchala, National Institute of Technology, Kurukshetra, India*
Jitender Kumar Chhabra, National Institute of Technology, Kurukshetra, India
Amit Rathee, Government College, Sonipat, India

ABSTRACT

Architecture recovery techniques study dependencies in source code and reconstruct architecture. Most techniques either use structural or semantic dependencies, and it is observed that the use of directory information helps in improving architecture recovery. The research carried out to date has focused on using the semantic information in a very limited manner and directory information in a trivial manner without considering directory hierarchy. Further, all three (structural, semantic, and directory-structure) are reported to be very useful in architecture recovery but have not been used in a combined manner at all. So, this paper proposes a new scheme for architecture recovery using a weighted combination of all three dependencies. A new approach is designed to effectively mine semantic dependencies and extract directory dependencies. Finally, different dependency schemes are evaluated with four clustering algorithms on three open-source projects. The obtained results show that the proposed scheme performs better than the existing approaches in architecture recovery.

KEYWORDS

Architecture Recovery, Directory Dependencies, Directory Hierarchy, Directory Structure, Integrated Dependencies, Semantic Dependencies, Structural Dependencies, Weighted Combination

INTRODUCTION

Software architecture is defined as the organization of a system embodying its components and their relationships. As software systems grow in size and complexity, it becomes hard for developers to keep architecture well-documented, and this phenomenon results in an architecture shift from its initial design. Most of the open-source projects lack architectural documentations and for these projects, code is the available documentation. So software architecture recovery is crucial for many reasons, to adapt a software system to changing requirements, to enable the reuse of components, and estimate the cost and risks involved in a change.

For this reason, huge research was carried out in this domain to recover the architecture of a software system, and architecture recovery is defined as a reverse engineering approach that aims

DOI: 10.4018/IJISMD.297060 *Corresponding Author

This article published as an Open Access article distributed under the terms of the Creative Commons Attribution License (http://creativecommons.org/licenses/by/4.0/) which permits unrestricted use, distribution, and production in any medium, provided the author of the original work and original publication source are properly credited.

at reconstructing architecture from the implementational view of software. Many techniques have already been proposed to recover the architecture of software and these techniques work on different types of input information. Depending on the input information used, these techniques are categorized as, structure-based techniques, semantic-based techniques, knowledge-based techniques (Kong et al., 2018). Structure-based techniques depend on the structure of source code to extract relations and group software elements based on structural dependencies using different clustering techniques. Semantic-based techniques depend on the textual information present in source code and documentation. These techniques try to form topics and group software elements into these topics. Knowledge-based techniques use various types of input information from software repositories viz; framework-related information, directory information, patterns, commits, and issues in version control systems.

In literature, the majority of architecture recovery techniques are either structure-based (Mancoridis et al., 1999) (Maqbool & Babri, 2004) (Andritsos & Tzerpos, 2005) (Wang et al., 2010) (Zhang et al., 2010) (Cho et al., 2019) or semantic-based (Kuhn et al., 2007) (Garcia et al., 2011) (Sajnani, 2012) (Link et al., 2019). Only a few techniques (Li et al., 2017) (Shahbazian et al., 2018) (Kong et al., 2018) (Guimaraes & Cai, 2020) exploit the available knowledge in software repositories and use them in architecture recovery. In software, readily available knowledge is its directory information, and only very few techniques (Kong et al., 2018) use this knowledge in architecture recovery. Most of the techniques use one or two types of input information in the recovery process. However, none of these techniques utilize structural, semantic, and directory information at the same time. Further, there is no proper study on how to extract available directory knowledge and integrate it with structural and semantic information for architecture recovery.

This paper aims to mine all needed semantic information, compute hierarchy-based directory dependencies information and integrate these with structural dependencies to recover the software architecture. Effective mining of semantic information including comments, identifiers, variables, class/method names as well as usage, is carried out and a new approach for extracting directory dependencies based on directory hierarchy is proposed. Various coupling schemes are formulated to evaluate the effect of using multiple dependencies in architecture recovery. These coupling schemes are also experimented with different sets of weights on three subject systems, to identify the best combination of weights for integrating dependencies. The main contributions of this paper include:

1. Designing a new approach for computing directory dependencies from directory hierarchy by using distance and depth-based measures.
2. Effective mining of all types of semantic information and empirically evaluation of the effect of using semantic and directory dependencies in architecture recovery by formulating six different dependency coupling schemes.
3. Integrating all three dependencies in the best combination of weights based on experimentation.
4. To study the effect of integrated dependencies in architecture recovery by using Complete linkage clustering.

RELATED WORK

In literature there are already various architecture recovery techniques, these techniques try to obtain dependency information from source code and group related software elements into modules to produce a high-level view of the system. In this study, the architecture recovery techniques are categorized into three types based on the input information they use in the recovery process.

Structure-Based Recovery Techniques

Bunch (Mancoridis et al., 1999) is a clustering tool that creates software decompositions automatically. It considers the module dependency graph and recovers architecture by optimizing the Modularization

Quality function on the graph. The weighted combined algorithm (WCA) (Maqbool & Babri, 2004) is a hierarchical agglomerative clustering technique. It calculates inter-cluster distance using Ellenberg measure and groups entities based on inter-cluster distance. Andritsos & Tzerpos (2005) proposed Scalable Information Bottleneck (LIMBO), a hierarchical clustering technique that employs an agglomerative information bottleneck algorithm for clustering. LIMBO uses information loss measures to group entities and it works on minimizing the information loss measure. Wang et al. (2010) proposed a fuzzy logic-based hierarchical clustering technique (LBFHC) which is an improvement over the LIMBO technique. Zhang et al. (2010) proposed a hybrid clustering algorithm HPCA based on partitional and hierarchical clustering. It considers a weighted directed class graph and uses hierarchical clustering to find kernels in the graph, then partitions the graph based on kernels. Cho et al. (2019) proposed an architecture recovery approach using cluster ensembles. It uses outputs of different clustering techniques and consolidates these outputs to recover a better architecture.

Semantic-Based Recovery Techniques

Kuhn et al. (2007) proposed an architecture recovery approach based on the information retrieval technique Latent semantic indexing (LSI), and it works by recognizing topics in source code and grouping similar artifacts into topics. Garcia et al. (2011) proposed a machine learning-based technique to identify connectors and components in a software system using software concerns. Sajnani (2012) proposed a machine learning-based automatic software architecture recovery approach. It considers domain, textual, structural, runtime behavioral, and contextual information. It uses unsupervised learning techniques for component identification and classification techniques to find utility and application components. Sun et al. (2017) proposed a novel program comprehension approach for clustering classes in large-sized packages using Latent Dirichlet Allocation (LDA). Recover and RELAX (Link et al., 2019) is a concern-oriented architecture recovery approach. It considers textual information from source code to extract concerns and uses a pre-trained classification model to relate entities to a specific domain.

Knowledge-Based Recovery Techniques

In this study, the architecture recovery techniques that use the information other than structural and semantic are categorized into knowledge-based recovery techniques. Li et al. (2017) proposed a framework information-based software architecture recovery approach. It considers framework-specific features and incorporates this information into clustering. Shahbazian et al. (2018) proposed RecovAr, an approach to recover the architectural design decisions. It links the issues found in version control systems to their corresponding code changes and identifies potential architectural changes. Kong et al. (2018) proposed a directory-based dependency approach for architecture recovery. It considers code dependencies between directories to generate a dependency graph and uses it in structure-based recovery techniques. Guimaraes & Cai (2020) proposed a language-independent pattern-oriented architecture recovery framework. It recovers design patterns and architectural design decisions from source code.

Evaluation Measures

Architectural decompositions that are obtained as a result of the recovery process must be assessed to evaluate their quality. This assessment can be carried out by using external and internal evaluation measures. In external evaluation, the recovered architecture is compared with an architecture created by experts. Precision and Recall (Anquetil & Lethbridge, 1999), are effectiveness measures that are based on intra pairs, where intra pairs are pairs of entities in the same cluster. Precision is defined as the percentage of intra pairs in the recovered architecture that are also in the expert architecture. Recall is defined as the percentage of intra pairs in the expert architecture that are also in the recovered architecture. MoJo (Tzerpos & Holt, 1999) is an external evaluation metric that measures the distance between two architecture decompositions by counting the number of move and join

operations needed in transforming one architecture to another. MoJoFM (Zhihua Wen & Tzerpos, 2004) is an effectiveness measure based on MoJo. EdgeMoJo (Zhihua Wen & Tzerpos, 2004) is an improvement of MoJo to account for edge weights. Obtaining expert architectures or constructing expert decompositions is difficult for most researchers, so they evaluate the recovered architecture based on the cohesion and coupling-ness between entities. Amarjeet & Chhabra (2017) and Rathee & Chhabra (2018) studied the measurement of coupling and cohesion between entities through the use of structural, semantic, and evolutionary dependencies. In architecture recovery, the majority of techniques used structural relations in computing cohesion and coupling. Turbo Modularization Quality (Turbo MQ) (Mitchell et al., 2001) (Mitchell, 2002) is one such measure that measures inter-connectivity and intra-connectivity in the recovered architecture. It is an internal quality measure, based on the assumption that recovered architecture should exhibit high cohesion and low coupling. MoJoFM and Turbo MQ measures are commonly used in the evaluation of architecture recovery.

Comparison of Architecture Recovery Techniques

In literature, there are many evaluation studies on architecture recovery techniques (Maqbool & Babri, 2007) (Bittencourt & Guerrero, 2009) (Shtern & Tzerpos, 2012) (Garcia et al., 2013) (Zahid et al., 2017). Maqbool & Babri (2007) showed that LIMBO performs better than WCA. Bittencourt & Guerrero (2009) showed that Modularization Quality and Design Structure Matrix techniques perform better in graph-based clustering. Shtern & Tzerpos (2012) presented the evaluation of software clustering approaches and outlined the major research challenges in this area. In (Garcia et al., 2013) it is shown that ACDC and Architecture Recovery using Concerns perform better than other techniques. These studies are not always consistent because they differ in clustering technique implementations, subject systems, and evaluation measures.

In literature, most of the architecture recovery techniques are either structure-based or semantic-based and consider only one specific type of input information in the recovery process. The research suggests that the accuracy of recovery techniques could be improved by using different types of input information. Only a few recovery techniques exploit directory information although it is the readily available architectural knowledge that exists for most of the systems. The research carried out to date has focused on using the semantic information in a very limited manner, and directory information in a trivial manner without considering hierarchy and path at all. Further, the three dependencies structural, semantic and directory-structure are reported to be very useful to recover the architecture but have not been used in a combined manner at all. These research gaps motivated the authors to propose a new approach for extracting directory dependencies based on directory hierarchy and mine all types of semantic information effectively to empirically evaluate the effect of all three (structural, semantic, and directory-structure) dependencies combined in architecture recovery.

PROPOSED APPROACH

Software architecture recovery is the process of reconstructing the architecture for software by studying its source code, and the recovered architecture helps in software maintenance and program comprehension. Architecture recovery techniques try to reconstruct the architecture by grouping related software elements into packages and sub-packages based on their structural, semantic, and directory relations.

In this paper, the first step in evaluating the effect of dependencies is to obtain structural, semantic, directory dependencies from the source code of the software. The effect of using directory dependencies and integrated dependencies in architecture recovery is studied from six different dependency coupling schemes, namely Structural (STR), Semantic (SEM), Structural + Semantic (STR+SEM), Structural + Directory (STR+DIR), Semantic + Directory (SEM+DIR), and Structural

+ Semantic + Directory (STR+SEM+DIR). This study helps us to determine that the use of directory dependencies in combination with structural or semantic dependencies improves the results. These results lead us to integrate all three dependencies and study their effect on architecture recovery. In this paper, each dependency coupling scheme is evaluated with four Hierarchical Agglomerative Clustering (HAC) techniques, namely Single linkage, Unweighted average linkage, Weighted average linkage, and Complete linkage. Finally, the results of these clusterings are analyzed with different evaluation metrics and each coupling scheme is experimented with different weights to learn the best possible weights for integrating dependencies.

Figure 1 diagrammatically shows the procedure adopted for evaluating different dependencies. During the whole evaluation process, the following five steps are followed, and finally, the results are analyzed.

Step 1: Obtaining structural, semantic, and directory-structure relations of software elements from source code, and these relations are modeled into dependencies between software elements later.

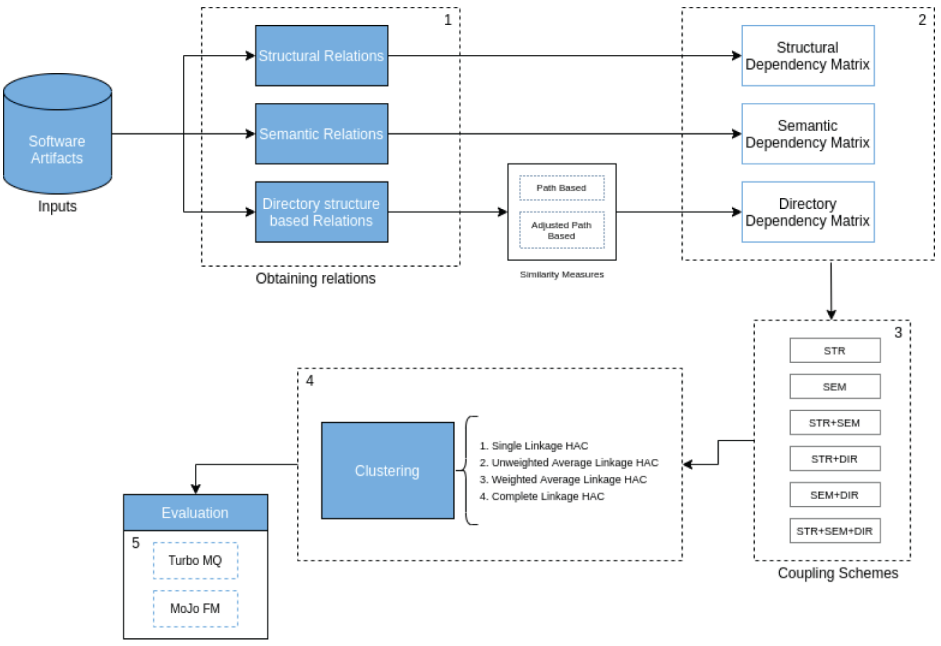
Step 2: Obtained relations are used in modeling dependencies and these dependencies are represented in the form of software dependency matrices. (Each element of the matrix represents the dependency strength between the corresponding pair of software elements.)

Step 3: The dependencies obtained are combined using six different coupling schemes, they are STR, SEM, STR+SEM, STR+DIR, SEM+DIR, STR+SEM+DIR and different weights are considered in integrating dependencies.

Step 4: Each coupling scheme is empirically evaluated with different clustering algorithms and the results are recorded.

Step 5: The obtained clustering results are evaluated using MoJoFM and Turbo MQ measures, the aim of performing this step is to find out how directory dependencies and integrated dependencies help in improving the overall architecture recovery.

Figure 1. Proposed approach to evaluate the effect of integrated dependencies in architecture recovery



Obtaining Structural Dependencies

Structural dependencies are obtained by analyzing the structural information present in the source code, such as references, method calls, parameters, etc. A previous study by (Muhammad et al., 2012) analyzed 26 types of structural relationships between entities in an object-oriented system and the relationships are listed in Table 1.

The file based and genericity based relationships are not relevant to the study because the dependency computation is made at the class level and only the static dependencies are considered. In this study, a more generalized relationship category is followed which incorporates all the structural relationships. Table 2 shows the eight relevant structural relations used in this study, these relations model most of the relations listed in Table 1 and also introduces few more important relations that exist in an object-oriented system. The number of structural inputs are greatly reduced using these eight relations so that clustering techniques can take advantage of these reduced and rich inputs. In (Prajapati et al., 2017; 2019; 2020) (Rathee & Chhabra, 2018; 2019) authors discussed the importance and proved the effectiveness of the eight relations listed in Table 2.

After identification of the relations, each relation is represented in the form of an n x n matrix, where n is the number of entities in the software system. Each entry in the matrix is based on the relationship between the corresponding entities. As an example, consider a hypothetical system with the following three classes C1, C2, and C3 as in Table 3. A '1' in the second row and fourth

Table 1. Structural relationships between entities in an object-oriented system

Inheritance based	*Containment based*	*Association based*
• Inheritance • same inheritance hierarchy • inheritance type • virtual method override • base class variable access • base class method access	• containment as objectsame class containment • variable accessmethod access	• maintaining pointer • maintaining reference • method parameter • method local • same class in methods
Global based	***Genericity based***	***File based***
• same global var. accesssame global func. accessame macro access	• instantiating parameter • same generic class • same generic parameter • inheritance via genericity • containment via genericity	• same file • include source file • same folder

Table 2. Structural relations under study

S. No.	Relation Name	Description
1	THROWS	Class A throws an exception of type Class B.
2	IS OF TYPE	Class A has an attribute or a field of type Class B.
3	CALLS	Method of Class A calls the method of Class B.
4	RETURNS	Class A has a method that returns an object of type Class B.
5	REFERENCE	Class A invokes an attribute or a method of Class B.
6	HAS PARAMETER	Class A has a method with a parameter of type Class B.
7	IMPLEMENTS	Class A implements the behavior specified by Class B.
8	EXTENDS	Class A extends Class B.

Table 3. n x n matrix showing EXTEND relations among entities

	C1	C2	C3
C1	0	0	1
C2	1	0	0
C3	0	0	0

column of the matrix represents that there is a relation between C1 and C3, for example, C1 may be inherited from C3.

Table 4 represents a hypothetical system with three classes C1, C2, and C3. A '2' in the second row and third column indicates that there is a relation between C1 and C2, for example, C1 calls the method of C2 and its frequency is 2.

To calculate structural dependencies based on structural relations, this paper uses the approach proposed by Rathee & Chhabra (2018). The relations listed in Table 2 are extracted by analyzing the source code and the relative weight (w_i) for each relation is calculated as:

$$Relative\ weight\left(w_i\right) = \frac{Total\ No.\ of\ i^{th}\ type\ of\ relations\ in\ the\ system}{Total\ No.\ of\ relations\ in\ the\ system} \tag{1}$$

The structural dependency between two classes C_i, C_j is calculated as:

$$StD\left(C_i, C_j\right) = \sum_{i=1}^{8}\left(f_i \times w_i\right) \tag{2}$$

Here f_i is the frequency of each relation and w_i is the relative weight of the i^{th} type of relation. The overall structural dependency matrix which contains structural dependencies between every pair of elements is calculated as:

$$StDM\left(i, j\right) = \sum_{i=1}^{|N|}\sum_{j=1}^{|N|} StD\left(C_i, C_j\right) \tag{3}$$

Here |N| is the total number of classes in the system and StDM is the overall structural dependency matrix.

Table 4. n x n matrix showing CALLS relations among entities

	C1	C2	C3
C1	0	2	0
C2	0	0	0
C3	0	3	0

Obtaining Semantic Dependencies

Software developed with proper coding standards and guidelines contains a fine amount of semantic information and the software elements that follow these standards, show a degree of semantic similarity (Rathee & Chhabra, 2017).

In this paper, semantic information of each class is obtained by mining the source code and documentation of software. Each source file is parsed to generate an abstract syntax tree and the abstract syntax trees are used in mining tokens from the comments, identifiers, class/method/variable definitions, and statements. After the mining process, each class is represented with a document containing a set of tokens. These documents are normalized and the following language preprocessing steps are applied to them, removal of programming language-specific keywords, removal of English language stop words, and applying of Porter's stemmer algorithm to stem words. In this paper, TF-IDF is used as a weighting scheme for tokens and it is calculated using the following formula:

$$tf - idf\left(t,d\right) = tf\left(t,d\right) \times \log\left(\frac{N}{df+1}\right) \tag{4}$$

Here, t ε T where T is the set of all tokens present in the system, d ε D where D = $\{d_1, d_2, d_3,.....,d_N\}$ is the set of all software elements, N = |D| is the total count of documents, tf is the term frequency and idf is the inverse document frequency. The semantic similarity between two classes C_i, C_j is calculated using cosine-based similarity measure as shown below:

$$Csim\left(C_i,C_j\right) = \frac{V_i.V_j}{V_i \times V_j} \tag{5}$$

$$Csim\left(C_i,C_j\right) = \frac{\sum_{k=1}^{n}\left(w_{k,i} \times w_{k,j}\right)}{\sqrt{\sum_{k=1}^{n} w_{k,i}^2}\sqrt{\sum_{k=1}^{n} w_{k,j}^2}} \tag{6}$$

Here, $V_i = (w_{i,1}, w_{i,2}, ...,w_{i,n})$ and $V_j = (w_{j,1}, w_{j,2}, ...,w_{j,n})$ is the vector representation of documents for the corresponding classes C_i, C_j respectively. The overall semantic dependency matrix (SmDM) is calculated as:

$$SmDM\left(i,j\right) = \sum_{i=1}^{|N|}\sum_{j=1}^{|N|} Csim\left(C_i,C_j\right) \tag{7}$$

Here |N| is the total number of documents. SmDM is symmetric and its values vary between [0, 1].

Obtaining Directory Dependencies

In literature, most of the techniques obtain information from code-level dependencies, which belongs to the implementational view of software. As software architecture belongs to logical view and there may be losing important logical information while translating information from implementation view to logical view (Kong et al., 2018).

The design of directories is one of the readily available logical information in software and most of the approaches ignore the available knowledge of directory structure in recovering the architecture. So

in this paper, to impart the available logical knowledge of directory structure into the recovery process, the path information of each class in the software system is extracted and used to generate a directory tree. To generate the path information each source file is examined for the classes defined in it and class names are appended with package names of the source files to generate the final path information. 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 by applying similarity measures based on the distance between nodes in the directory tree. Most of the similarity measures based on distance would degenerate to trivial measures when applied on a tree (Sologub, 2011), so to compute similarity the following two measures are used:

Distance-Based Measure

Distance is an opposite concept of similarity and it can be used to construct similarity measures. The standard similarity measure based on distance is expressed as:

$$S\left(C_i, C_j\right) = \frac{1}{1 + D\left(C_i, C_j\right)} \tag{8}$$

Here, $D(C_i, C_j)$ is the length of path/distance between tree nodes of classes C_i, C_j. In obtaining dependencies based on the directory tree the above measure shows trivial results because little importance is given to the hierarchical arrangement of classes in the directory tree. Consider an example directory tree as shown in Figure 2. The following pairs (C_1, C_2) and (C_4, C_5) show similarity of 0.33 and 0.33 on applying the usual distance-based similarity measure, but in the designer's logical view, the similarity of (C_1, C_2) should be more than the similarity of (C_4, C_5) because the nodes C_1, C_2 are more specialized than C_4, C_5 in the directory hierarchy.

Distance-Depth Based Measure

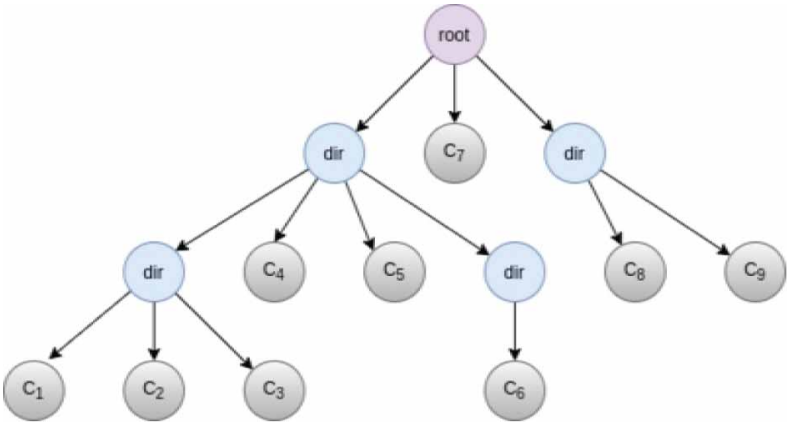
Generally, while designing, developers tend to create file hierarchies to group related files and directories. So in developers' view files at greater depth in the hierarchy are more related than the files at a lower depth. To reduce the effect of trivial distance-based measures and account for granularity, depth is used in similarity computation and it is calculated as the distance between the root node and common ancestor node (Sologub, 2011). The distance-depth based measure for similarity computation is expressed as:

$$S_D\left(C_i, C_j\right) = \frac{1}{1 + \dfrac{D\left(C_i, C_j\right)}{D\left(A_{i,j}, R\right)}} \tag{9}$$

$$S_D\left(C_i, C_j\right) = \frac{1 + D\left(A_{i,j}, R\right)}{1 + D\left(A_{i,j}, R\right) + D\left(C_i, C_j\right)} \tag{10}$$

Here, $A_{i,j}$ is the lowest common ancestor of (C_i, C_j) and R is the root. Using the example in Figure 2, the pairs (C_1, C_2), (C_4, C_5) show similarities of 0.6, 0.5 respectively on applying distance-depth based measure. This is because of the depth used in similarity computation which allows considering the directory structure hierarchy to produce nontrivial results. By using equation (10) the directory dependency matrix (DrDM) is calculated as:

Figure 2. Example directory tree



$$DrDM\left(i,j\right)=\sum_{i=1}^{|N|}\sum_{j=1}^{|N|}S_{D}\left(C_{i},C_{j}\right) \tag{11}$$

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

Integration of Dependencies and Clustering

In this study, a custom tool is designed to extract structural, semantic, and directory dependencies from the java systems, and with little modification, this custom tool can be made adapted to software systems written in other object-oriented languages. After obtaining structural, semantic, and directory structure relations and their corresponding dependency matrices, six different coupling schemes are used in integrating dependencies viz; STR, SEM, STR+SEM, STR+DIR, SEM+DIR, and STR+SEM+DIR. These coupling schemes are formulated by considering dependencies as individuals and as a weighted combination of dependencies. Each coupling scheme is experimented with different weights to learn the best combination of weights to integrate dependencies.

To study the effect of integrated dependencies in architecture recovery, each coupling scheme is evaluated with four clustering algorithms. This study uses Single linkage, Unweighted average linkage, Weighted average linkage, and Complete linkage hierarchical agglomerative clustering algorithms, and the results of applying these clustering algorithms are recorded.

Evaluation

In this paper, the recovered architecture is evaluated with MoJoFM (External assessment) and Turbo MQ (Internal assessment) measures. MoJoFM measures the similarity of recovered architecture to the expert architecture. Turbo MQ measures the cohesiveness of clusters formed in the recovery process and it is independent of expert architectures.

EXPERIMENTS AND RESULTS

The main aim of this study is to find how directory dependencies in combination with semantic or structural dependencies perform in architecture recovery, how does the combination of all three dependencies affects the results of recovery, and how directory dependencies from directory

hierarchy are computed by using distance and depth based measures. This work answers the following research questions:

RQ1: Can directory dependencies combined with structural and semantic dependencies improve the accuracy of a recovery technique?
RQ2: How does the distance-depth measure perform in obtaining directory dependencies when compared to the usual distance measure?
RQ3: How does the proposed integrated dependences approach perform compared to the approaches in literature?

Experimental Setup

Subject Projects

In this study, three open-source projects ArchStudio, Hadoop, and Tiny-Weka are selected to investigate the three research questions. Table 5 shows detailed information about subject systems. The selected projects are mainly Java because of its strong package mechanism, which is closely related to the directory structure. Lutellier et al. (2015) constructed expert architectures for Chromium, ITK, Bash, Hadoop, ArchStudio software and open-sourced their work. Two of the subject systems Hadoop and ArchStudio used in this study are selected from Lutellier et al. (2015) work and also many previous studies (Garcia et al., 2013) (Lutellier et al., 2015) (Kong et al., 2018) used these projects for architecture recovery evaluation. The third subject system is Tiny-Weka which is selected randomly and its expert architecture is constructed manually.

Accuracy Measures

To evaluate the results of a recovery technique there are two different types of measurements in literature, one measures the internal quality of the recovered architecture, and the other measures the similarity between recovered and expert architectures. In this paper, Turbo MQ (Mitchell et al., 2001) (Mitchell, 2002) is used for internal quality assessment and MoJoFM (Zhihua Wen & Tzerpos, 2004) is used for external evaluation.

Turbo Modularization Quality (Turbo MQ)

Turbo MQ is an internal quality measure that is based on the assumption that recovered architecture should exhibit low coupling and high cohesion. To calculate Turbo MQ, the Cluster Factor needs to be calculated first. The Cluster Factor for module i is calculated as:

$$CF_i = \frac{2\mu_i}{2\mu_i + \sum_j \left(\varepsilon_{ij}, \varepsilon_{ji}\right)} \tag{12}$$

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. Turbo MQ is defined as the sum of all cluster factors and it is calculated by the following formula:

Table 5. Subject systems details

Project	Version	Language	Description	Total Classes
ArchStudio	4	Java	Architecture Development	582
Tiny-Weka	3.9	Java	Data Mining Tool	339
Hadoop	0.19.0	Java	Data Processing	590

$$Turbo\,MQ = \sum_{i=1}^{k} CF_i \tag{13}$$

MoJoFM

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

$$MoJoFM = \left(1 - \frac{mno\left(P,Q\right)}{\max\left(mno\left(\forall P,Q\right)\right)}\right) \times 100\% \tag{14}$$

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

RESULTS AND DISCUSSION

This section presents the results obtained by evaluating six dependency coupling schemes on three subject systems. The clustering results obtained by applying four algorithms are analyzed and it is observed that the Complete Linkage HAC algorithm outperforms all others and forms more cohesive clusters. Anquetil & Lethbridge (1999) and Maqbool & Babri (2007) also described in their work that, the results of applying the Complete Linkage HAC are better when compared to Single Linkage HAC, Unweighted Average Linkage HAC and Weighted Average Linkage HAC algorithms. So in this paper, only the results of applying Complete Linkage HAC are demonstrated and the evaluation results of the other clusterings are presented in the work (Puchala, 2021). Table 6 demonstrates the overall scores of MoJoFM and Turbo MQ obtained by evaluating each coupling scheme with the Complete Linkage HAC on three subject systems.

In this paper, the hit and trial approach is used in determining the best combination of weights for each coupling scheme. The first three columns of Table 6 lists different weight combinations used in this study for integrating structural, semantic, and directory dependencies. The experimentation results in this study indicate that the following weight combinations (0.5, 0.5); (0.6, 0.4) for pairwise coupling schemes and (0.4, 0.2, 0.4) for STR+SEM+DIR coupling scheme results in best scores. In Table 6 the best scores of each coupling scheme, when applied to a particular subject system, are highlighted in light gray.

RQ1: Can directory dependencies combined with structural and semantic dependencies improve the accuracy of a recovery technique?

Figure 3 and figure 4 show a comparison of various coupling schemes based on their best scores obtained in the evaluation of three subject systems using MoJoFM and Turbo MQ. It is clear from the plots that using pairwise dependency schemes in architecture recovery produces more accurate results. Based on scores it is observed that the integration of directory and structural dependencies in recovery results in an average improvement of 32% in scores of MoJoFM and an adequate improvement in the scores of Turbo MQ when compared to the scores of using structural dependencies alone. By comparing the scores of using semantic dependencies and scores of integrating semantic and directory dependencies, an average improvement of 15% is observed in the scores of MoJoFM. To leverage most of the information present in software repositories and to form better cohesive

Table 6. Obtained Turbo MQ and MoJoFM scores on three subject systems

Weights			ArchStudio		Hadoop		Tiny-Weka	
STR	SEM	DIR	Turbo MQ	MoJoFM	Turbo MQ	MoJoFM	Turbo MQ	MoJoFM
1	0	0	9.8	32.86	6.6	16.43	1.4	68.18
0	1	0	13.5	51.87	13.1	41.7	2.6	67.88
0.3	0.7	0	21.4	55.24	19.9	44.52	3.6	69.39
0.4	0.6	0	21.8	56.48	20.4	43.11	2.8	66.97
0.5	0.5	0	21.8	55.6	20.9	44.88	3.6	69.7
0.6	0.4	0	21.7	58.08	20.7	43.29	2.9	67.27
0.7	0.3	0	23.8	59.33	20.8	42.93	2.9	71.52
0.9	0	0.1	22	68.03	18.8	50.71	3.5	87.27
0.8	0	0.2	20.9	69.45	19.2	52.83	3.3	84.24
0.7	0	0.3	20.2	70.69	19.3	55.65	3.6	86.97
0.6	0	0.4	21.7	69.98	19.7	53.89	3.7	91.52
0	0.9	0.1	15.3	58.61	12.9	44.35	3	72.73
0	0.8	0.2	15.2	60.92	13.5	46.11	3	86.67
0	0.7	0.3	16.6	62.52	14.8	51.41	3.6	86.67
0	0.6	0.4	13.9	67.67	14.9	51.77	3.7	89.7
0.5	0.3	0.2	23	71.4	21.4	50	3.8	83.33
0.4	0.4	0.2	22.9	70.52	22.7	51.06	3.4	82.73
0.3	0.5	0.2	19.3	64.65	21.4	49.82	3	81.52
0.5	0.2	0.3	23	74.96	19.4	54.42	3.9	90.91
0.4	0.3	0.3	22.2	71.58	19.9	53.36	3.5	83.33
0.3	0.4	0.3	22.5	71.4	20.7	52.65	3.7	83.64
0.2	0.5	0.3	20.5	68.92	21.1	52.65	2.9	81.21
0.4	0.2	0.4	23.8	75.49	20.2	54.42	4	93.33
0.3	0.3	0.4	23.3	74.07	19.9	54.59	3.8	90
0.2	0.4	0.4	22.2	71.4	21.3	53	3.7	85.76

clusters, the dependency coupling scheme STR+SEM+DIR is proposed. This scheme integrates all three structural, semantic, and directory dependencies and it is observed that this coupling scheme outperforms all other schemes, and when the results of this coupling scheme are compared with the STR+SEM coupling scheme, an average improvement of 17% in the scores of MoJoFM is observed and a fine improvement in the scores of Turbo MQ. The subject system Hadoop shows only a small variation in the scores of Turbo MQ and MoJoFM because of its directory hierarchy, Garcia et al. (2013) showed that this version of Hadoop has fewer directories with many components scattered.

It is clear from figures 3 and 4 that imparting existing directory knowledge into the recovery process by pairing it with structural or semantic dependencies improves the accuracy of a recovery technique and the proposed dependency coupling scheme STR+SEM+DIR which is defined as a weighted combination of structural, semantic and directory dependencies outperforms all other coupling schemes earlier formulated in this paper.

Figure 3. Comparison of coupling schemes based on MoJoFM scores

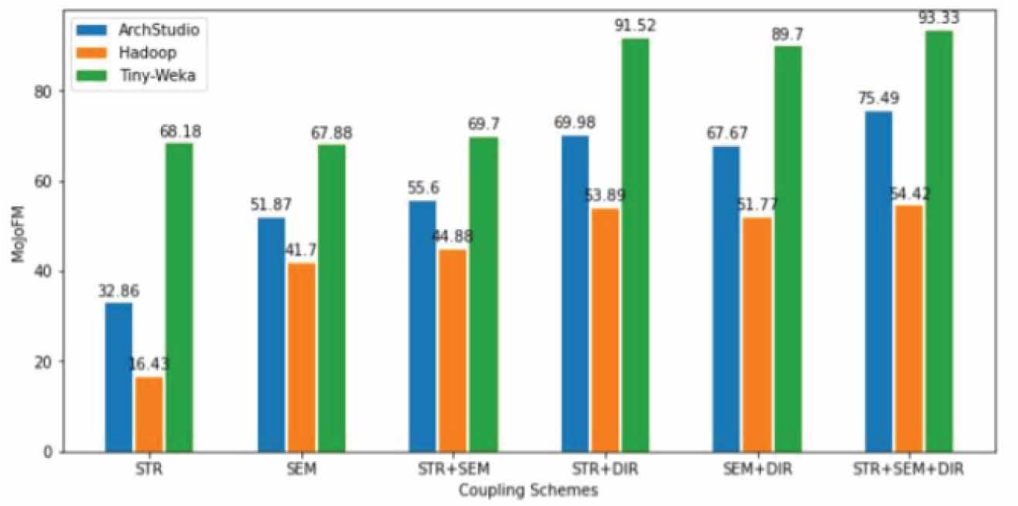
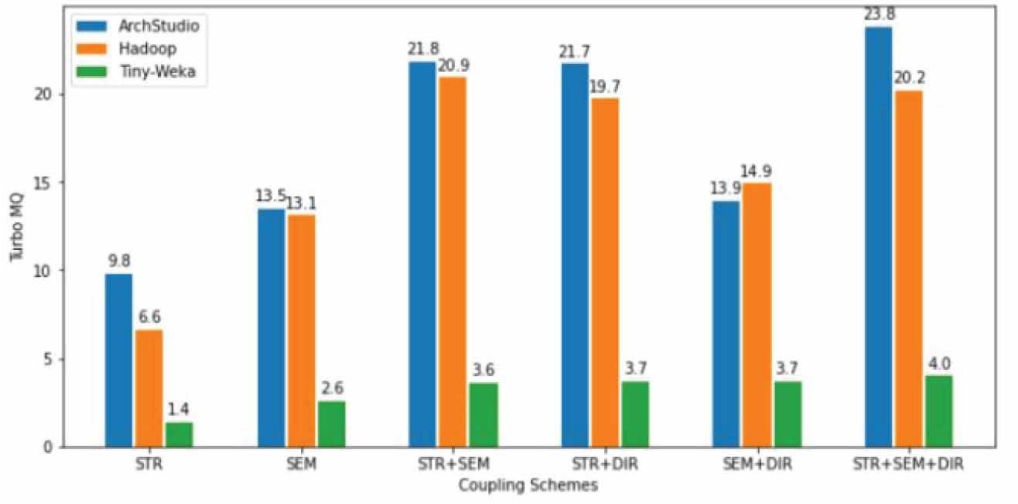


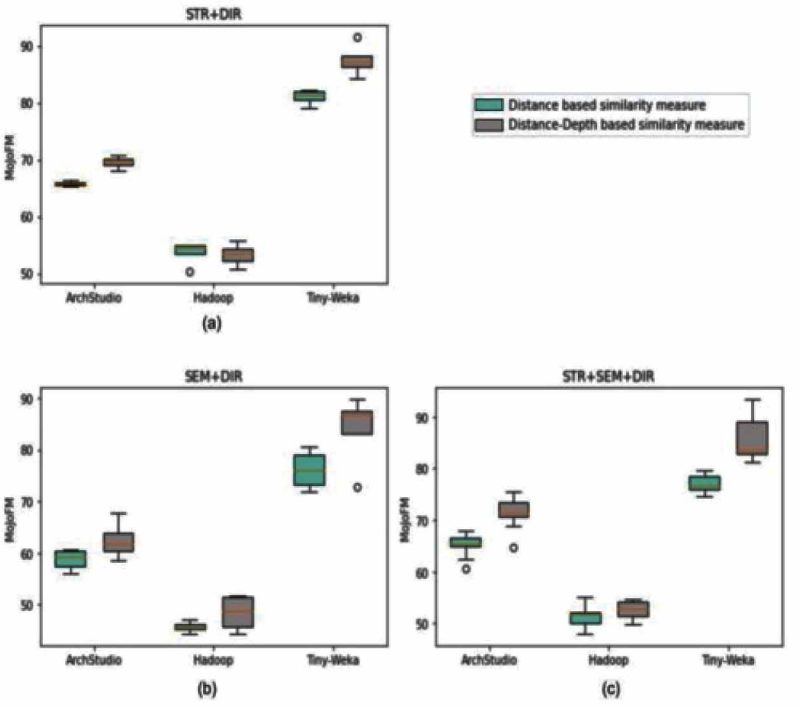
Figure 4. Comparison of coupling schemes based on Turbo MQ scores



RQ2: How does the distance-depth measure perform in obtaining directory dependencies when compared to the usual distance measure?

To test RQ2, the evaluation scores of each coupling scheme are recorded by considering distance and distance-depth measures in similarity computation of directory dependencies. The boxplots in Figure 5 are plotted based on MoJoFM scores obtained by using distance and distance-depth measures. The boxplots show high variability in the scores of MoJoFM using the distance-depth measure. From the boxplots (a), (b), (c) in Figure 5 it is clear that the distance-depth measure produces much better results when compared to the usual distance measure. It is observed that on all three subject systems and in each coupling scheme that involves directory dependencies by using distance-depth measure

Figure 5. Comparison of distance and distance-depth based similarity measures



produces required nontrivial results. Based on the obtained scores of MoJoFM on three subject systems, it is concluded that the distance-depth measure captures directory dependencies perfectly by using depth and performs better in architecture recovery.

RQ3: How does the proposed integrated dependences approach perform compared to the approaches in literature?

To show how integrated dependencies improve the architecture recovery results, a comparison is provided between Kong et al. (2018) and the proposed approach. Kong et al. (2018) proposed approach generates a submodule-level dependency graph based on directory hierarchy and in this approach directory dependencies are used together with structural dependencies to generate the final architecture. Bunch-NAHC, Bunch-SAHC techniques from (Mancoridis et al., 1999) are used for clustering in Kong et al. (2018) and the obtained results are presented separately for each of these clustering approaches. The MoJoFM and Turbo MQ approaches are further used to evaluate obtained recovered architecture. Therefore, in order to efficiently evaluate and compare the approach of Kong et al. (2018) with the proposed approach of this paper, the individual scores of these two clusterings approaches are averaged and compared with the scores obtained using the proposed approach. Table 7 presents the scores of MoJoFM and Turbo MQ obtained after evaluating the proposed approach and the values presented by Kong et al.'s (2018) approach on Hadoop and ArchStudio. The main reason behind considering Hadoop and ArchStudio is that these software systems are evaluated by Kong et al. (2018) and their proposed approach is not publicly available for experimentation purposes.

It can be observed from Table 7 that there is an overall improvement of 8-20% in the scores of MoJoFM. The MoJoFM scores show how close the recovered architecture is to the expert's constructed architecture. The improved score for MoJoFM indicates the fact that the architecture recovered by

Table 7. Comparison of the proposed approach with Kong et al.'s (2018)

Subject Systems	Kong et al.'s (2018)		Proposed Approach	
	MoJoFM	Turbo MQ	MoJoFM	Turbo MQ
ArchStudio	62%	26.35	75%	23.8
Hadoop	49%	18.1	54%	19.9

the proposed approach is much closer to the experts' architecture when compared to the architecture recovered by Kong et al. (2018). Hence, it can be easily claimed that the proposed approach of this paper sustains its accuracy while recovering software architecture. Moreover, the obtained Turbo MQ score shows a minor improvement in the case of Hadoop software. Whereas, for ArchStudio, the Turbo MQ score shows some decrease to the value as observed in Kong et al. (2018). However, these values are comparable. A higher value of Turbo MQ represents better architectural design in terms of cohesion and coupling quality parameters. Based on obtained results as presented in Table 7 and their interpretations it can be concluded that the proposed approach in this paper is capable of recovering high-quality architectural design having sufficiently high modularization quality. Hence, it is clear from the results that the proposed approach outperforms the Kong et al. (2018) approach because of the effective utilization of dependencies. Finally, it can be concluded that a better usage of integrated dependencies improves the overall architecture recovery results.

THREATS TO VALIDITY

The following four threats to the validity of the proposed approach are identified. The first threat to validity is the directory hierarchy of software, for the subject system Hadoop, it is observed that there is only a slight improvement in recovery results because it has fewer directories with many components scattered. So it is observed that the quality of directory hierarchy impacts the results of the proposed approach. The second threat is regarding the determination of weights used in integrating different dependencies. Preferably, all combinations of weights are to be considered, but it is not feasible. Thus, the weights in this study are determined using the hit and trial approach and all the major possibilities have been considered to attenuate this threat. The third threat to validity is the clustering techniques used for architecture recovery. The proposed methodology operates on four basic hierarchical clustering algorithms and many techniques are specifically tailored for architecture recovery and produce more accurate results. To reduce this threat more experiments are to be conducted on more subject systems with more clustering techniques. The fourth threat to validity is the measures involved in the computation of cohesion and coupling. In literature a variety of information such as structural details, the semantics of source code, and also component evolution history is used in computing cohesion and coupling, these different approaches need to be investigated to find their effect in architecture recovery. In this study, the threat is minimized by considering structural, semantic, and directory relations, and their effect is studied by considering the individual, pairwise, and triplet in architecture recovery.

CONCLUSION AND FUTURE WORKS

Software architecture is crucial for effective maintenance, as it helps in program comprehension significantly. After some iterations of changes and growth of the software, maintaining a well-documented architecture becomes difficult due to various drifts in the architecture. So, architecture recovery techniques have been proposed to recover the architecture of software by mining its source code and other documents. The accuracy of an architecture recovery technique mainly depends

on the input information used. This paper proposes an effective approach of obtaining directory dependencies from the directory hierarchy of software and mining the suitable semantic information for collecting more meaningful inputs for architecture recovery. In this paper, the impact of using different dependencies in architecture recovery is studied by using six dependency coupling schemes. These coupling schemes are designed to evaluate the effect of using structural, semantic, directory dependencies in architecture recovery, and a new scheme is proposed based on the integration of all three dependencies. The best weights for combining the three dependencies are identified by experimentation on three open-source projects. Based on the study it is observed that using only a single type of dependency is not enough and the use of integrated dependencies shows a significant improvement in the results of overall architecture recovery.

The software systems that lack proper architectural documentations are hard to maintain, in this setting the results of the proposed approach can be used as architectural descriptions and also can help in studying the system components and connections. If the current architecture of a system deviates from planned architecture then it is not easy to understand software even with documentation support. In such cases, the recovery results of the proposed approach can be used to study current architecture and take necessary actions. Open source projects are known for their lack of documentation support, in this scenario the results of the proposed approach can be used in getting a high-level understanding of the project.

In the future, the proposed approach can be investigated by considering different sources of information in modeling the relations between entities. The proposed integrated dependencies approach can be utilized for software restructuring, remodularization, and fault prediction. There is also a scope that the proposed approach of directory dependencies based on directory hierarchy can be used in the construction of ground truth architectures. Moreover, the weights used in the combination of all three dependencies are learned by experimentation, but a new approach can be designed where weights could be learned automatically based on the quality of input information.

REFERENCES

Amarjeet & Chhabra, J. (2017). Improving modular structure of software system using structural and lexical dependency. *Information And Software Technology, 82*, 96-120.

Andritsos, P., & Tzerpos, V. (2005). Information-theoretic software clustering. *IEEE Transactions on Software Engineering*, *31*(2), 150–165. doi:10.1109/TSE.2005.25

Anquetil, N., & Lethbridge, T. (1999). Experiments with clustering as a software remodularization method. *Sixth Working Conference On Reverse Engineering* (Cat. No.PR00303), 235-255. doi:10.1109/WCRE.1999.806964

Bittencourt, R., & Guerrero, D. (2009). Comparison of Graph Clustering Algorithms for Recovering Software Architecture Module Views. *2009 13th European Conference on Software Maintenance and Reengineering*.

Cho, C., Lee, K., Lee, M., & Lee, C. (2019). Software Architecture Module-View Recovery Using Cluster Ensembles. *IEEE Access: Practical Innovations, Open Solutions*, *7*, 72872–72884. doi:10.1109/ACCESS.2019.2920427

Garcia, J., Ivkovic, I., & Medvidovic, N. (2013). A comparative analysis of software architecture recovery techniques. *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*.

Garcia, J., Krka, I., Mattmann, C., & Medvidovic, N. (2013). Obtaining ground-truth software architectures. *2013 35th International Conference on Software Engineering (ICSE)*.

Garcia, J., Popescu, D., Mattmann, C., Medvidovic, N., & Cai, Y. (2011). Enhancing architectural recovery using concerns. *2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)*.

Guimaraes, E., & Cai, Y. (2020). Understanding Software Systems through Interactive Pattern Detection. *2020 IEEE International Conference On Software Architecture Companion (ICSA-C)*.

Kong, X., Li, B., Wang, L., & Wu, W. (2018). Directory-Based Dependency Processing for Software Architecture Recovery. *IEEE Access: Practical Innovations, Open Solutions*, *6*, 52321–52335. doi:10.1109/ACCESS.2018.2870118

Kuhn, A., Ducasse, S., & Gîrba, T. (2007). Semantic clustering: Identifying topics in source code. *Information and Software Technology*, *49*(3), 230–243. doi:10.1016/j.infsof.2006.10.017

Li, X., Zhang, L., & Ge, N. (2017). *Framework Information Based Java Software Architecture Recovery. In 2017 24th Asia-Pacific Software Engineering Conference Workshops*. APSECW.

Link, D., Behnamghader, P., Moazeni, R., & Boehm, B. (2019). Recover and RELAX: Concern-Oriented Software Architecture Recovery for Systems Development and Maintenance. *2019 IEEE/ACM International Conference On Software And System Processes (ICSSP)*. doi:10.1109/ICSSP.2019.00018

Lutellier, T., Chollak, D., Garcia, J., Tan, L., Rayside, D., Medvidovic, N., & Kroeger, R. (2015). Comparing Software Architecture Recovery Techniques Using Accurate Dependencies. *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*.

Mancoridis, S., Mitchell, B., Chen, Y., & Gansner, E. (1999). Bunch: a clustering tool for the recovery and maintenance of software system structures. *Proceedings IEEE International Conference On Software Maintenance - 1999 (ICSM'99). 'Software Maintenance For Business Change'* (Cat. No.99CB36360). doi:10.1109/ICSM.1999.792498

Maqbool, O., & Babri, H. (2004). The weighted combined algorithm: a linkage algorithm for software clustering. *Eighth European Conference On Software Maintenance And Reengineering, 2004. CSMR 2004. Proceedings*, 15-24. doi:10.1109/CSMR.2004.1281402

Maqbool, O., & Babri, H. (2007). Hierarchical Clustering for Software Architecture Recovery. *IEEE Transactions on Software Engineering*, *33*(11), 759–780. doi:10.1109/TSE.2007.70732

Mitchell, B. (2002). *A Heuristic Search Approach to Solving the Software Clustering Problem (Ph.D.)*. Drexel University.

Mitchell, B., Traverso, M., & Mancoridis, S. (2001). An architecture for distributing the computation of software clustering algorithms. *Proceedings Working IEEE/IFIP Conference On Software Architecture*. doi:10.1109/WICSA.2001.948427

Muhammad, S., Maqbool, O., & Abbasi, A. (2012). Evaluating relationship categories for clustering object-oriented software systems. *IET Software*, *6*(3), 260. doi:10.1049/iet-sen.2011.0061

Prajapati, A., & Chhabra, J. (2019). Information-Theoretic Remodularization of Object-Oriented Software Systems. *Information Systems Frontiers*, *22*(4), 863–880. doi:10.1007/s10796-019-09897-y

Prajapati, A., Parashar, A., & Chhabra, J. (2020). Restructuring Object-Oriented Software Systems Using Various Aspects of Class Information. *Arabian Journal for Science and Engineering*, *45*(12), 10433–10457. doi:10.1007/s13369-020-04785-z

Puchala, S. (2021). *Software Architecture Recovery Using Structural, Semantic and Directory Features* [Unpublished master's thesis]. National Institute of Technology, Kurukshetra.

Rathee, A., & Chhabra, J. (2017). Software Remodularization by Estimating Structural and Conceptual Relations Among Classes and Using Hierarchical Clustering. *Communications in Computer and Information Science*, *712*, 94–106. doi:10.1007/978-981-10-5780-9_9

Rathee, A., & Chhabra, J. (2018). Clustering for Software Remodularization by Using Structural, Conceptual, and Evolutionary Features. *Journal of Universal Computer Science*, *24*(12), 1731–1757.

Rathee, A., & Chhabra, J. (2019). Mining Reusable Software Components from Object-Oriented Source Code using Discrete PSO and Modeling Them as Java Beans. *Information Systems Frontiers*, *22*(6), 1519–1537. doi:10.1007/s10796-019-09948-4

Sajnani, H. (2012). Automatic software architecture recovery: A machine learning approach. *2012 20th IEEE International Conference On Program Comprehension (ICPC)*.

Shahbazian, A., Kyu Lee, Y., Le, D., Brun, Y., & Medvidovic, N. (2018). Recovering Architectural Design Decisions. *2018 IEEE International Conference On Software Architecture (ICSA)*. doi:10.1109/ICSA.2018.00019

Shtern, M., & Tzerpos, V. (2012). Clustering Methodologies for Software Engineering. *Advances in Software Engineering*, *2012*, 1–18. doi:10.1155/2012/792024

Sologub, G. (2011). On Measuring of Similarity between Tree Nodes. In *Proceedings of the Fifth Russian Young Scientists Conference in Information Retrieval* (pp. 63-71). St. Petersburg University Press.

Sun, X., Liu, X., Li, B., Li, B., Lo, D., & Liao, L. (2017). Clustering Classes in Packages for Program Comprehension. *Scientific Programming*, *2017*, 1–15. doi:10.1155/2017/3787053

Tzerpos, V., & Holt, R. (1999). MoJo: a distance metric for software clusterings. *Sixth Working Conference On Reverse Engineering* (Cat. No.PR00303). doi:10.1109/WCRE.1999.806959

Tzerpos, V., & Holt, R. (2000). ACCD: an algorithm for comprehension-driven clustering. *Proceedings Seventh Working Conference On Reverse Engineering*, 258-267. doi:10.1109/WCRE.2000.891477

Wang, Y., Liu, P., Guo, H., Li, H., & Chen, X. (2010). Improved Hierarchical Clustering Algorithm for Software Architecture Recovery. *2010 International Conference On Intelligent Computing And Cognitive Informatics*. doi:10.1109/ICICCI.2010.45

Wen, Z., & Tzerpos, V. (2004). An effectiveness measure for software clustering algorithms. *Proceedings of the 12th IEEE International Workshop on Program Comprehension*, 194-203.

Wen, Z., & Tzerpos, V. (2004). Evaluating similarity measures for software decompositions. *20th IEEE International Conference on Software Maintenance Proceedings*.

Zahid, M., Mehmmod, Z., & Inayat, I. (2017). Evolution in software architecture recovery techniques — A survey. *2017 13th International Conference On Emerging Technologies (ICET)*.

Zhang, Q., Qiu, D., Tian, Q., & Sun, L. (2010). Object-oriented software architecture recovery using a new hybrid clustering algorithm. *2010 Seventh International Conference On Fuzzy Systems And Knowledge Discovery*. doi:10.1109/FSKD.2010.5569799

Shiva Prasad Reddy Puchala is currently pursuing his Master's degree in Computer Engineering from National Institute of Technology, Kurukshetra, India. He received his Bachelor's degree in Computer Science and Engineering from Nalla Malla Reddy Engineering College, Hyderabad, Telangana, India, in 2018.

Jitender Kumar Chhabra is Professor, Dept of Computer Engg, National Institute of Technology, Kurukshetra India. He has been always topper throughout his career. and has more than 28 years of teaching & research experience. He has published more than 140 papers in reputed journals and conferences. He is author of three international books and filed 8 patents out of which five have been published. Additionally, four copyrights for software have been granted to him as Intellectual Property Rights (IPR). He has completed a Research Project by DRDO, Govt of India. He is Reviewer for IEEE Transactions, ACM Transactions, Elsevier, Springer, Wiley, IGI Global & many other journals. He has guided six scholars for their PhD and three are in progress. One of his guided PhD theses has been selected as resource Material by ACM, USA. His areas of interest are Software Engineering, Clustering, Soft Computing and Object-Oriented Systems. Earned his PhD from NIT Kurukshetra.