Stronger Together: On Combining Relationships in Architectural Recovery Approaches

Evelien Boerstra*
Univ. of British Columbia, Canada
boerste@student.ubc.ca

John Ahn*
Univ. of British Columbia, Canada
jahn18@student.ubc.ca

Julia Rubin
Univ. of British Columbia, Canada
mjulia@ece.ubc.ca

***Abstract*—Architecture recovery is the process of obtaining the intended architecture of a software system by analyzing its implementation. Most existing architectural recovery approaches rely on extracting information about relationships between code entities and then use the extracted information to group closely related entities together. The approaches differ by the type of relationships they consider, e.g., method calls, data dependencies, and class name similarity. Prior work shows that combining multiple types of relationships during the recovery process is often beneficial as it leads to a better result than the one obtained by using the relationships individually. Yet, most, if not all, academic and industrial architecture recovery approaches simply unify the combined relationships to produce a more complete representation of the analyzed systems. In this paper, we propose and evaluate an alternative approach to combining information derived from multiple relationships, which is based on identifying agreements/disagreements between relationship types. We discuss advantages and disadvantages of both approaches and provide suggestions for future research in this area.**

***Index Terms*—architecture recovery, software re-engineering, relationships between code entities.**

I. Introduction

Software architecture recovery techniques [27], [32], [43]–[46], [50], [66], [71] aim at extracting architectural information from lower-level software representations, such as source code. They assist software developers and architects by augmenting often outdated or even non-existing architectural documentation with up-to-date information which is in full sync with the implemented system. Closely related to the field of architectural recovery are microservice extraction techniques, which recently gathered substantial attention in both academia and industry [13], [18], [24], [28], [29], [37], [38], [47], [48], [52], [56], [58], [61], [69]. These techniques are designed to help developers migrate legacy applications from monolithic to the microservice architecture style [42], effectively splitting a monolithic software system into a set of interdependent modules (a.k.a. microservice candidates).

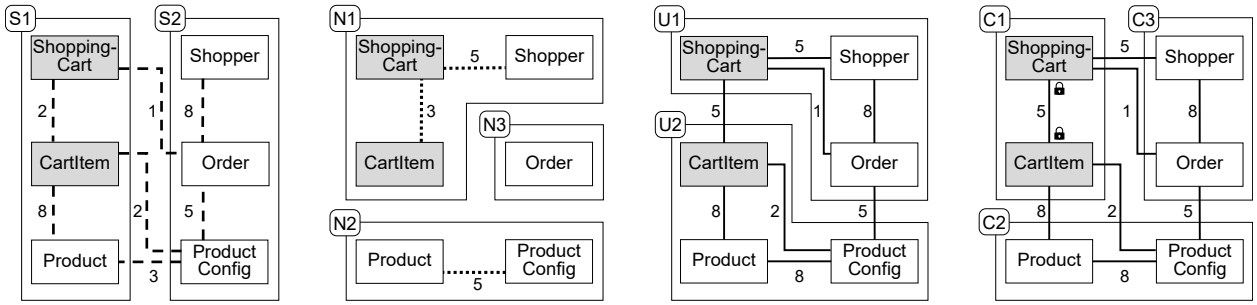
Most of these techniques share the same underlying principle: they construct a graph representation of the analyzed software system, in which the nodes represent application elements, e.g., packages, classes, methods, etc., and the edges represent (weighted) relationships between the elements, e.g., statically or dynamically collected method calls and data dependencies, name similarities, evolutionary similarities, etc. Once such a graph-based representation is constructed, the techniques utilize existing [30] or proprietary [50], [71] clustering algorithms to group nodes into separate partitions (a.k.a. clusters), which represent architectural modules. The main objective of clustering is to achieve loose coupling, i.e., minimizing inter-cluster connections, and high cohesion, i.e., maximizing intra-cluster connections, w.r.t. the considered relationships.

Most of the existing techniques operate at a class- or file-level granularity, which was shown to align with users' expectations the best [19], [22], [23]. The techniques vary by the types of relationships between elements they consider during the decomposition process. Moreover, existing work shows that considering only one type of relationship is often insufficient [44] and, thus, most of the existing approaches combine multiple types of relationships, e.g., method calls and class hierarchies [28], method calls and class name similarities [37], statically and dynamically extracted method call info [58], structural relationships and evolutionary data about artifact co-changes [29], and more.

These approaches typically use (what we refer to as) a *union-based strategy* for combining information from multiple relationships. That is, for each pair of elements, they unify information from different sources into one combined edge between the elements. Consider, for example, a simplified version of an online shopping application in Figure 1, which is inspired by one of our case studies. This example application consists of six classes: *Shopper*, which represents a user of the system; *Order*, which represents the shopping order of the user and includes the details about the price of the order, payment methods, and shipping details; *ShoppingCart*, which aggregates *CartItems* the user selects for their order; *Product* that contains detailed information about a product and its possible variations; and, finally, *ProductConfig* which defines particular product configurations selected by the user.

Figure 1a shows the decomposition produced when considering (weighted) static method call relationships between the classes. Such an approach is often taken to capture the architectural structure of the system. This decomposition consists of two clusters: *S1*, which contains the *ShoppingCart*, *CartItem*, and *Product* classes, and *S2*, which contains the *Shopper*, *Order*, and *ProductConfig* classes. The *Product* class has a higher affinity with *CartItem* than with *ProductConfig* because *CartItem* often queries the *Product* class to present accurate product information to the user. At the same time, *ProductConfig* has a higher affinity with *Order* than with *Product* because the *Order* class needs to access various aspects

*Equal contribution.



(a) By Static Relationships. (b) By Name Similarities. (c) Union-based. (d) Concensus-based.

Fig. 1: Examples of Possible Application Decompositions.

of the product configuration to identify the price, shipment options, etc. Thus, these two classes were split into two different clusters in this decomposition.

Figure 1b shows a different decomposition that was obtained by considering the similarity between class names. This approach is often taken to capture business domains encoded in element names. The produced decomposition contains three clusters: *N1* groups *Shopper*, *ShoppingCart*, and *CartItem* together, as both *Shopper* and *CartItem* have names similar with *ShoppingCart*. *N2* groups *Product* and *ProductConfig* together, due to the similarity in their names. *Order* does not have an edge connecting it with other classes as their names are not similar; it is thus placed in a separate cluster, *N3*.

A typical union-based approach for combining static and name-based relationships from Figures 1a and 1b is shown in Figure 1c. It aggregates all edges between each pair of classes and sums up their weights. For example, the edge between the *Product* and *ProductConfig* classes has a weight of 8 in this example as it combines the static relationship with a weight of 3 and the name-based similarity relationship with a weight of 5. In practice, edge weights are typically normalized during such a union, as we discuss in the next section, but we omitted this in our example for simplicity of the demonstration.

The decomposition produced by these unified edges contains two clusters, *U1* and *U2*, which represent cohesive elements w.r.t. both relationships combined. However, even though the *ShoppingCart* and *CartItem* classes (shaded in the figure) were placed in the same cluster in both by-static and by-name-similarity decompositions, there are split apart between different clusters in this clustering result.

In this paper, we investigate an alternative approach for combining information derived from multiple relationships, which relies on the intuition that elements which were grouped together in both individual decompositions are likely to belong together. We refer to our approach, which is inspired by the clustering ensemble work from the field of machine learning and data science [67], as a *consensus-based strategy*. In this strategy, we first compute decompositions by each individual relationship type and then "lock together" elements that are part of the same cluster in each individual decomposition, like the classes *ShoppingCart* and *CartItem* in our example. We then complete the clustering for the remaining elements using a union of relationships.

Figure 1d shows an example of such consensus-based decomposition. As *ShoppingCart* and *CartItem* are locked together, they are placed in their own cluster, *C1*. To obtain the best result w.r.t. the couplings and cohesion for the remaining elements, *Product* and *ProductConfig* are placed in *C2*, while *Shopper* and *Order* are placed in *C3*. That is, in fact, the expected result in this example, as it correctly separates shopping cart, product, and user details.

To investigate the applicability and usefulness of both strategies in practice, we conduct an empirical study with six large-scale real-life applications. We picked these case studies because they include manually constructed and curated expected decomposition results (a.k.a. ground truth) and are commonly used for evaluating architectural recovery and microservice decomposition techniques [44], [63], [65], [70].

We decompose the applications using both union- and consensus-based strategies and use similarity metrics to compare the obtained and expected results. Our analysis shows that the consensus-based strategy outperforms the union-based strategy in the majority of cases, producing a result closer to the users' expectations.

Contributions. This paper makes the following contributions.
1. It defines the problem of combining multiple relationship types in architectural recovery and microservice extraction.
2. It introduces a new consensus-based strategy for combining information derived from multiple relationships and empirically compares it with the union-based strategy used in prior work on six large-scale case study applications. The results of the comparison show that the consensus-based strategy outperforms union-based in the majority of cases.
3. It makes our implementation, empirical evaluation setup, and evaluation results publicly available to facilitate reproducibility and future work in this area [14].

The remainder of the paper is structured as follows. Section II formally defines both union- and consensus-based strategies for performing decompositions using multiple relationships. In Section III, we discuss our instantiation of the approaches, including our selections of relationships and clustering techniques. Section IV discussed our study methodology and Section V presents the results of our evaluation, lessons learned, and future research directions. Limitations and threats to validity of our study are discussed in Section VI, Section VII outlines the related work and, finally, Section VIII concludes the paper.

II. Strategies for Combining Relationships

In this section, we formally define the *union-based strategy* for combining relationships, which is used in prior work [19], [37], [49], [57]. We then define the *consensus-based strategy* we introduce in this work.

We represent a software system as a graph (N, R), where the set of nodes N corresponds to elements of the system, such as classes, methods, files, or packages, and a set of undirected weighted edges R corresponds to the relationships between the elements. For the example in Figure 1, N is the set of classes *ShoppingCart*, *CartItem*, *Product*, *Shopper*, *Order*, and *ProductConfig*. There are two types of relationships in this example: R_1, which corresponds to the static relationships in Figure 1a and R_2, which corresponds to the name-based similarities in Figure 1b.

Each relationship $r \in R$ has a weight, denoted by $w(r)$. To fairly combine the relationships without one "dominating" over the other, we ensure that the weights are normalized and are ranging between 0 and 1, i.e., $\forall r \in R, w(r) \in [0, 1]$.

We define a *decomposition* D of a graph (N, R) as a partition of N into a set of *clusters* $\{c_1, \ldots, c_k\}$. Each cluster $c_i \in D$ is a non-empty subset of elements from N, i.e., $c_i \subseteq N \wedge c_i \neq \emptyset$. We require all clusters to be mutually exclusive, $c_i \cap c_j = \emptyset$, and cover the entire set N, $\cup_{i=1}^{k} c_i = N$. In other words, every element of N must belong to one and only one cluster.

A. Union-based Strategy

The union-based strategy, which we refer to as $\mathbb{U}$, assumes a software system with two types of relationships, i.e., graphs (N, R_1) and (N, R_2). It combines the information from the different relationship types and derives a unified representation $(N, \bar{R})$, which consists of the same set of nodes N and a new set of edges $\bar{R}$. There exists an edge $e \in \bar{R}$ between two nodes $n_i, n_j \in N$ if and only if there is an edge e_1 between these nodes in R_1 and/or an edge e_2 between the nodes in R_2. We define the weight of the edge $w(e) = \frac{w(e_1)+w(e_2)}{2}$, assuming that $w(e_i) = 0$ if $e_i \notin R_i$.

For the example in Figure 1c, we summed up rather than averaged weights of the combined relationship for illustration purposes. We create an edge between *Product* and *ProductConfig* with a weight of 8 as it combines the weights of the static and by-name similarity edges in Figures 1a and 1b, with weights 3 and 5, respectively. The edge between the *ShoppingCart* and *Shopper* elements has a weight of 5, like in the by-name similarity relationship graph, as no such edge exists in the static relationship graph.

The unified graph $(N, \bar{R})$ is then used to cluster the nodes towards producing the union-based decomposition $D^{\mathbb{U}}$, as shown in Figure 1c of our example.

B. Consensus-based Strategy

The consensus-based strategy, which we refer to as $\mathbb{C}$, preserves elements grouped together in decompositions of N produced using individual relationships R_1 and R_2. The intuition behind this strategy is to preserve the consensus between the decompositions induced by different relationship types when reconciling relationships.

More formally, given decompositions D_1 and D_2 over (N, R_1) and (N, R_2), respectively, we first find all *consensus groups* $\hat{c}_{ij}$ between D_1 and D_2:

$$\hat{c}_{ij} = c_i \cap c_j \mid c_i \in D_1, c_j \in D_2$$

We denote by $\hat{C}$ the set of all non-empty consensus groups, i.e., all groups of elements that were clustered together in both decompositions:

$$\hat{C} = \left\{ \bigcup_{i \in [1,|D_1|],\ j \in [1,|D_2|]} \hat{c}_{ij} \mid \hat{c}_{ij} \neq \emptyset \right\}$$

For the example in Figure 1d, there is one non-empty consensus group consisting of the classes *ShoppingCart* and *CartItem* as these classes were grouped together in both static and by-name similarity decompositions. This consensus group is indicated by lock symbols in Figure 1d. Note that each element can appear in at most one consensus group as clusters in each decomposition are mutually exclusive.

We define a new consensus graph $(N, L, \bar{R})$, where N are nodes representing the elements of the system, $\bar{R}$ are edges between the nodes, defined as in the union-based strategy, and L is the set of labels annotating the nodes. L contains a distinct label for each consensus group in $\hat{C}$, $|L| = |\hat{C}|$. We denote the label of a node n in $(N, L, \bar{R})$ by $l(n)$ and ensure that nodes share the same label if and only if they belong to the same consensus group:

$$\forall n_i, n_j \in N, l(n_i) = l(n_j) \iff \exists \hat{c} \in \hat{C} \mid n_i \in \hat{c} \ \wedge \ n_j \in \hat{c}$$

We say that a label of a node n is empty if n does not belong to any consensus group: $l(n) = \emptyset$ if $\nexists \hat{c} \in \hat{C} \mid n \in \hat{c}$.

The consensus graph $(N, L, \bar{R})$ is, again, used to cluster elements in N while ensuring that all elements with the same label are always placed in the same resulting cluster. We refer to the produced decomposition as the consensus-based decomposition $D^{\mathbb{C}}$. Figure 1d shows an example of such a decomposition for our motivating example.

III. Instantiation of the Approach

In this section, we discuss the relationships and the clustering algorithms that we use to instantiate and empirically compare the identified strategies.

A. Selected Relationship Types

Prior work on architectural recovery and microservice extraction techniques considered a variety of relationship types between application elements [27], [32], [39], [44], with the most common being structural, semantic, and evolutionary relationships. Structural relationships aim to capture architectural similarity between application elements and group together elements that are likely to belong to the same architectural component. Semantic relationships aim to capture lexical similarity between application elements and group together elements that use the same terminology. The idea behind this type of relationships is that elements using similar terminology are likely to belong to the same application domain. Finally,

evolutionary relationships aim to capture the structure of the team working on an application and group together elements developed by the same team members. The idea here is to capture elements developed by independent sub-teams, as they can represent independent components.

For our experiments, we instantiated the approach with two types of relationships: structural and semantic. In particular, we experimented with class/file-level static and name-based similarity relationships, which we describe below.

Static Relationships. These relationships represent control and data dependencies between elements. A control dependency occurs when one class/file invokes a method from another class/file. A data dependency occurs when a class/file uses a data structure defined in another class/file as its field, local variable, method parameter, or return variable. In the graph representing static relationships, these dependencies are translated into weighted undirected edges, where the weight is set to the sum of all control and data dependencies between the corresponding connected elements, in both directions.

For the example in Figure 1a, the edge between the *ShoppingCart* and *CartItem* classes indicates that there exist two control and/or data dependencies between these two classes. In this case, *ShoppingCart* contains a field of type *CartItem* and invokes a method in the *CartItem* class. The *CartItem* does not invoke any methods or use data of type *ShoppingCart*. Thus, the weight of the edge is set to 2.

For our experiments, we extracted static relationships using the Understand by SciTools [11] tool. We set edge weights by the number of static dependencies (or "references" in SciTools terminology), which include the aforementioned dependencies.

Name-based Similarity Relationships. These relationships capture the similarity of class/file names. Intuitively, they aim to identify elements that belong to the same domain by relying on naming conventions followed by developers. There exists a name-based similarity relationship between two elements if the elements contain similar terms in their name. For the example in Figure 1b, *ShoppingCart* and *CartItem* share an edge because of the shared term "Cart" in their names.

We rely on classic Information Retrieval (IR) techniques, including name tokenization, lemmatization, and the removal of stopwords to calculate these relationships. First, all class/file names are tokenized according to the naming convention of the application (e.g., camelCase, PascalCase, Train-Case or Snake_Case) into a set of terms. Each term is lemmatized and stop words are removed from the set. Stop words include the most common English words [5] and keywords of the Java and C/C++ programming languages.

We then calculate the degree of similarity between two names as the fraction of the number of common terms used in both names out of the number of all unique terms used in both names. That is, let the name X be defined by $terms(X)$ and the name Y be defined by $terms(Y)$. The name similarity between name X and Y is then defined as

$$similarity(X, Y) = \frac{|terms(X) \cap terms(Y)|}{|terms(X) \cup terms(Y)|}$$

For example, *ShoppingCart* and *CartItem* in Figure 1 are represented in PascalCase. They are tokenized as *{Shopping, Cart}* and *{Cart, Item}*, respectively. Lemmatization reduces the term *Shopping* to *Shop*. In this case, no stopwords are present. The names share one common term, *Cart*. The similarity between the names is then $\frac{1}{3}$ (in Figure 1b, we multiplied all numbers by 10 for demonstration purposes).

For our experiments, we consider the file basename (i.e., the rightmost segment of the file path) of an application element as the name of that element. Additionally, we used the Word Net Lemmatizer offered by the Natural Language Toolkit [7] to extract and lemmatize tokens.

Relationship Normalization. In our case, the weight of static relationships is unbounded and is represented by whole numbers, whereas the weight of name relationships is bounded in the range of 0 to 1. To fairly combine relationships whose weights are not within the same range, weights must first be normalized; otherwise, one relationship type can dominate another and the overall decomposition would largely resemble the one with the more dominant relationship weights.

We perform normalization by first standard-normalizing all edge weights in the edge set R and then transforming them to values between 0 to 1. That is, for each edge e with the weight $w(e)$, we define the z-score normalized edge weight $z(w(e))$ as:

$$z(w(e)) = \frac{w(e) - \mu_R}{\sigma_R}$$

where μ_R represents the mean and σ_R represents the standard deviation of the edge set R. Shifting the edge set distribution to the standard normal distribution does not affect the shape or spread of the distribution, but rather centers it at 0. In other words, the distribution adopts a mean of 0 and standard deviation of 1. This ensures that z-score normalized edge weights from different distributions are comparable.

As z-scores measure the number of standard deviations an edge weight is from the edge set mean, they are sensitive to outliers [36] and do not produce weights in the common 0 to 1 range. That could bias the clustering in favor of an edge set with a greater distribution spread. We thus transform each z-score normalized edge weight $z(w(e))$ by the standard logistic function, sigmoid curve, defined by $s(x) = 1/(1 + \exp(-x))$. This function has a domain of all real numbers. Its return value ensures that edges with large z-score normalized weights, either positive or negative, are assigned weights closer to 0 and 1, respectively, whereas weights near the mean for each distribution are assigned weights near 0.5. That is, the function does not change the distribution shape but adjusts the spread so that edges have a weight between 0 and 1, reducing the impact of outliers and increasing the contribution of edges near the distribution mean [72].

B. Clustering Algorithms

Graph clustering is a known problem in computer science with many applications in software engineering, social networks, biology, medicine, and more. In particular, there are several clustering techniques developed specifically for architecture recovery problems [17], [25], [27], [32], [45], [50],

as well as a large number of generic, off-the-shelf clustering techniques [35], [60], [71]. To ensure our results are not specific to a particular clustering technique, we picked for our experiments one specific and one generic clustering approach, as described below.

BUNCH [50] is one of the most popular open source clustering algorithms designed for software modularization; it is commonly used in in architectural recovery studies [43], [44], [59], [70]. Its objective function, Turbo Modularization Quality (*TurboMQ*), optimizes for high cohesion and low coupling w.r.t. intra- and inter-relationships of software elements grouped into clusters. More specifically, given a decomposition of elements into k clusters, a *cluster factor* CF_i for each cluster i is calculated as $CF_i = \frac{\mu_i}{\mu_i + 0.5 \times \sum_j \epsilon_{ij} + \epsilon_{ji}}$

where μ_i is the number of intra-relationships and $\epsilon_{ij} + \epsilon_{ji}$ is the number of inter-relationships between cluster i and any other cluster j. *TurboMQ* is then defined as the sum of *cluster factors* of all clusters of a decomposition:

$$\textit{TurboMQ} = \sum_{i=1}^{k} CF_i$$

BUNCH provides a number of optimization algorithms: Nearest Ascent Hill Climbing (NAHC), Steepest Ascent Hill Climbing (SAHC), and a genetic algorithm. Prior work found that, although the genetic algorithm finds a solution more quickly, the hill-climbing algorithms produce higher-quality results, with NAHC having a lower tendency to timeout than the SAHC approach [44], [62]. Therefore, we used the NAHC version in our analysis. We configured BUNCH to perform 500 random initializations for each clustering problem (an initial population size of 500) and try all possible moves in each run (a hill climb percentage of 100%).

Spectral clustering [53] is a technique rooted in graph theory; it is used for solving a relaxation of an NP-hard discrete graph partitioning problem [26], in particular, to identify clusters in a weighted graph whose nodes correspond to data points and edges represent distances between the points. As such, it is very suitable for our task: finding communities of nodes in a graph based on the edges connecting them.

We used an off-the-shelf spectral graph clustering technique implemented by the scikit-learn python library [55], particularly the *cluster.SpectralClustering* module. As spectral clustering assumes the target number of clusters k as the input, we instantiated it twice: (i) by setting k to be equal to the number of clusters in the expected architecture for each subject application and (ii) by setting k to be equal to the number of clusters produced by BUNCH. This was done to investigate the effect of the number of clusters on the quality of decompositions produced by the consensus-based approach.

Implementing Decomposition Approaches. To produce decompositions by individual relationship types, as well as the union decomposition $D^{\mathbb{U}}$, we simply inputted the corresponding graphs to BUNCH and the spectral clustering approaches.

To produce the consensus decomposition, $D^{\mathbb{C}}$, we first calculated consensus groups from the individual decompositions, as described in Section II. We leveraged and augmented the open source implementation of BUNCH to ensure that consensus groups stay together during clustering. Specifically, we ensured that all elements in a consensus group are assigned to the same cluster during initialization and are also moved together in any subsequent move that the tool explores. As our approach does not alter the clustered elements and their relationships, it does not interfere with BUNCH's *TurboMQ* objective function, providing a fair comparison with the union approach.

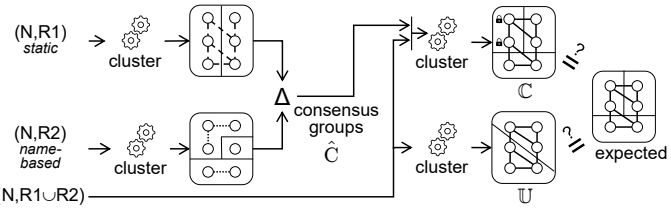


Fig. 2: Evaluation Overview.

For the spectral clustering, we followed the advice on StackOverflow [64], contracting the input adjacency metric. Specifically, we replaced all elements belonging to a consensus group C with a single element c and created an edge between c and each element v that was adjacent to an element of C. We set the weight of an edge between c and v to be the maximal weight of all elements of C that were adjacent to v, to preserve the strongest relationship to v. We treated contracted elements separately when computing all similarity metrics, for fair comparison with the union-based approach.

For the example in Figure 1, we merged the *ShoppingCart* and *CartItem* classes of the consensus group into one element, which has four outgoing edges: to *Shopper*, *Order*, *Product*, and *ProductConfig*. As each of these classes has only one edge connecting it to an element of the consensus group, the maximum of edge weights is trivially equal to the original weight of such edge.

IV. EVALUATION SETUP

In this section, we describe our experimental methodology and evaluation setup. Our investigation is driven by the following research questions:

RQ1 (Usefulness) Is the consensus-based strategy plausible?

- **RQ1.1** What fraction of consensus-based groups is preserved in the expected decomposition?
- **RQ1.2** What fraction of consensus-based groups is split in the union-based decomposition?

RQ2 (Performance) How well does the consensus-based strategy perform compared with the union-based strategy?

- **RQ2.1** How close are both decompositions to the expected result?
- **RQ2.2** What is the execution time for both decompositions?

Figure 2 outlines our overall evaluation approach. We used individual relationships to decompose each subject software system and also produced union- and consensus-based decompositions $D^{\mathbb{U}}$ and $D^{\mathbb{C}}$, respectively. We compared these decompositions to the expected result using a set of metrics.

In what follows, we discuss our selection of subject applications and their expected decompositions, the metrics we used, and our runtime evaluation environment.

A. Subject Applications

We selected six large open-source projects commonly used in evaluation of architectural recovery techniques [15], [37], [44], [63], [65], [70] as subject applications for our work (see Table I). These projects span a number of application domains, are implemented in three different programming languages (C, C++, Java), two different programming paradigms (procedural and object-oriented), and greatly vary in size.

Three of the projects – Bash, Hadoop, and ArchStudio are borrowed from Lutellier et al. [44]. We selected these projects as they included manually curated expected decomposition results (a.k.a. ground-truth architectures) the authors obtained through collaboration with engineers closely familiar with these systems [4]. We excluded ITK and Chromium from our study as, at the time of writing, we could not retrieve the version of the projects used by the authors.

In addition, we followed the approach by Teymourian et al. [70] and Jin et al. [37] and included Mozilla Firefox and XWiki projects as the authors confirmed that the desired application decomposition conforms with the application package structure in these cases.

Finally, we augmented our set of projects with a popular Java web application, PartsUnlimited, used in prior work on decomposing monolithic to microservice-based architecture [15], [39] and which also has an open-source ground-truth version. Next, we describe these subject systems in more detail.

Bash [3] (the Bourne-Again SHell) is a command-line shell that provides a user interface to a GNU operating system (OS). Bash is written in C and is included in popular OSs, such as GNU/Linux and Mac OS X. Version 4.2 of the system is implemented in 364 .c and .h files comprising 102 KLOC. The ground-truth architecture splits these files into 14 clusters, which contain all the original files and 9 duplicated files.

Hadoop [1] is a widely used open-source framework for distributed data processing in compute clusters [10]. Following prior studies [44], we focused on the HDFS, Map-Reduce, and Core components of Hadoop version 0.19.0, as we have access to the ground-truth architecture for these components. The components are implemented in Java, consist of 90 KLOC, and contain 615 Java classes. The ground-truth architecture splits this implementation into 67 clusters, which contain all the original classes and 157 duplicates.

ArchStudio [2] is a development environment for modeling, analyzing, implementing, and visualizing systems and software architectures. ArchStudio version 4 is implemented in Java, consists of 58 KLOC, and contains 583 Java files. Its ground-truth architecture contains 57 clusters, which contain all the original files, without any duplicates.

Mozilla Firefox [6] is an open-source web browser written in C++. We followed the approach by Teymourian et al. [70] and used ten out of 55 main folders of the developer preview version 3.7a4, release 1.9.3a4, as there is a credible human expert confirmation that sub-folders of these ten folders can act as the decomposition ground truth. The selected folders are: Accessible, Browser, Build, Content, Db, Dom, Extensions, Gfx, Intl and Ipc. Excluding tests, they contain 3437 .cpp, .c and .h files comprising 600 KLOC, and include 67 sub-folders we used as the ground truth. Unlike prior work that analyzed each folder individually, we focused our analysis on the entire selected code, to perform a system-level analysis. Our method thus considers all the files of the selected folders together in a flat structure and assesses the similarity of the produced clusters to the existing sub-folder structure.

TABLE I: Subject Applications

Case study	Project version	Lang.	KLOC	# Classes / Files	# Clusters in Expected
Bash	4.2	C	102	364	14
Hadoop	0.19.0	Java	90	615	67
ArchStudio	4	Java	58	583	57
Mozilla Firefox	1.9.3a4	C++	600	3437	67
XWiki	14.1	Java	264	3135	76
PartsUnlimited	-	Java	3	52	5

XWiki [12] is an open source wiki software platform written in Java with a design emphasis on extensibility. For our analysis, we use the most recent stable version with a verified signature on GitHub, namely version 14.1. We focused on the xwiki-platform-core component, which is implemented in 3135 Java files consisting of 264 KLOC. As with Mozilla Firefox, we used the existing directory structure, which contains 58 sub-folders, as the ground-truth architecture.

PartsUnlimited [8], [9] is a Java-based open source Manufacturing Resource Planning application which was developed for training purposes and contains both monolithic and microservice-based versions of the backend sub-system. We focused on decomposing the monolithic version in commit a83586b, which contains 3 KLOC in 52 Java files. The microservice-based version of the backend contains five clusters, with all the original files and 28 duplicates.

B. Metrics

To assess the usefulness of consensus-based groups and to answer **RQ1**, we measure the fraction of the groups that are indeed *preserved* in the expected decomposition for each case study. That is, given the expected decomposition $D^{\mathbb{E}}$ and a set of consensus groups $\hat{C}$, we compute a subset of consensus groups $\hat{C}^P \subseteq \hat{C}$ *preserved* in the expected decomposition, such that each $\hat{c}^P \in \hat{C}^P$ is a subset of at least one cluster $c^E \in D^{\mathbb{E}}$, i.e., $\exists c^E \in D^{\mathbb{E}} \ |\hat{c}^P \subseteq c^E$. We report on the fraction of preserved consensus groups out of the total number of consensus groups, $\frac{|\hat{C}^P|}{|\hat{C}|}$.

While all consensus groups $\hat{C}$ are, by definition, preserved in the consensus-based decomposition, we also assess how these groups are handled during the union-based decomposition. To this end, we calculate and report the number of consensus groups $\hat{C}^{S_U} \subseteq \hat{C}$ that are *split* (i.e., not preserved) by the union-based decomposition. That is, given a decomposition $D^{\mathbb{U}}$, $\hat{C}^{S_U}$ includes all consensus groups in $\hat{C}$ that are *not* a subset of any of the clusters in $D^{\mathbb{U}}$. We report both the total number

of such consensus groups and the number of consensus groups in the "important" subset of $\hat{C}^{S_U}$ – those that are preserved in the expected decomposition, i.e., $\hat{C}^{S_U} \cap \hat{C}^{P}$.

To answer **RQ2**, we assess each produced decomposition based on its closeness to the expected results using a set of common architecture recovery metrics [19], [20], [33], [40], [41], [43], [44], [54], which we describe below.

MoJoFM [73] is a non-symmetric metric that takes as input two clustered architectures, A and B, and quantifies the number of *Move* and *Join* operations required to transform architecture A into B. The *Move* operation moves an entity from one cluster to another existing or a newly created cluster. The *Join* operation joins two clusters into one and reduces the number of clusters by one. *MoJoFM* assigns the same weight to both operations. More formally, *MoJoFM* is defined as:

$$\textit{MoJoFM}(A,B) = \left(1 - \frac{mno(A,B)}{\max(mno(\forall A,B))}\right) \times 100\%$$

where $mno(A,B)$ is the minimum number of operations needed to transform architecture A into B and $\max(mno(\forall A,B))$ is the maximum number of non-repeated steps to transform any architecture A (with the same elements) into B. Placing $\max(mno(\forall A,B))$ in the denominator effectively 'scales' the value of *Move* and *Join* operations to match the size of its input architectures.

MoJoFM scores range from 0% to 100%, wherein a higher value represents a higher similarity between two architectures. In our analysis, we measure the value of transforming a produced architecture into the expected architectures, as our goal is to assess how 'architecturally distant' the produced architectures are from the ground-truth architectures.

Architecture-to-architecture (a2a) [40] is a metric proposed to address the main drawback of *MoJoFM*: that its *Join* operation is excessively cheap for clusters containing a high number of elements. This is particularly visible for large produces, resulting in high *MoJoFM* values for architectures with many small clusters. Moreover, *a2a* handles discrepancies between the recovered and ground-truth architectures, i.e., when the recovered architecture contains a higher/lower number of elements [40]. *a2a* is formally defined as:

$$a2a(A,B) = \left(1 - \frac{mto(A,B)}{aco(A) + aco(B)}\right) \times 100\%$$

$$mto(A,B) = addE(A,B) + remE(A,B) + movE(A,B) + addC(A,B) + remC(A,B)$$

$$aco(X) = addC(X_\emptyset, X) + addE(X_\emptyset, X) + movE(X_\emptyset, X)$$

where $mto(A,B)$ is the minimum number of operations needed to transform architecture A into B and $aco(X)$ is the number of operations needed to construct any architecture X from a "null" architecture $X_\emptyset$. mto and aco calculate the total number of the five operations used to transform one architecture into another: $addE$ handles cases where elements exist in the ground-truth architecture B but not in A, by "symbolically" adding them to A; $remE$ removes elements from A if they do not exist in B, $movE$ moves an element from one cluster of A to another, ensuring that it is placed with elements that appear in the same cluster in B. At the cluster level, $addC$ adds a new cluster in A and $remC$ removes one.

Cluster-to-cluster Coverage (c2c$_{cvg}$) [33], [43], [44] aims to assess cluster-level accuracy by measuring the degree of overlap between the clusters of two architectures:

$$c2c(c_i, c_j) = \frac{|c_i \cap c_j|}{\max(|c_i|, |c_j|)} \times 100\%$$

where c_i is a cluster in the produced architecture A and c_j is a cluster in the ground-truth architecture B. The denominator is used to normalize the entity overlap in the numerator by the number of entities in the larger of the two clusters, ensuring that *c2c* provides the most conservative value of similarity between the two clusters.

To summarize the extent to which clusters of a decomposition match ground-truth clusters, *c2c$_{cvg}$* calculates the fraction of clusters that are th_{cvg}-similar to at least one cluster in the ground truth, where th_{cvg} is a threshold that can be set to a certain percentage, e.g., 10% or 50%. More formally,

$$c2c_{cvg}(A,B) = \frac{|simC(A,B)|}{|B|} \times 100\%$$
$$simC(A,B) = \{c_i | (c_i \in A | \exists c_j \in B) \wedge c2c(c_i, c_j) > th_{cvg})\}$$

Similar to prior work [44], we assessed the cluster-level accuracy using three different thresholds: 50% (high similarity), 33% (moderate similarity), and 10% (some similarity).

C. Runtime Environment

We used five virtual machines (VM) running in a proprietary compute cluster consisting of three physical machines with Intel Xeon E5-2640 v4 @ 2.40GHz and Intel Xeon E5-2680 v4 @ 2.40GHz processors. Each VM runs an Ubuntu 16.04 operating system and has 4 cores and 16 GB of RAM.

V. Results

We now describe the results of our evaluation for each of the research questions. We further summarize lessons learned from our experience and outline suggestions for future research.

A. RQ1: Usefulness

Table II shows the total number of consensus groups ($\hat{C}$), the number and fraction of the consensus groups that are also preserved in the expected architecture ($\hat{C}^{P}$), and the number and fraction of the consensus groups that are split by the union-based decomposition ($\hat{C}^{S_U}$). We present this information for each case study application and for BUNCH and spectral clustering algorithms, separately. Furthermore, for spectral clustering, we present the results with k set to the number of clusters in the ground-truth architecture and k set to the number of clusters produced by BUNCH.

The table shows that a large fraction of consensus groups – 86%, 63.7%, and 75.5% on average for BUNCH, *Spectral (k=GT)*, and *Spectral (k=Bunch)*, respectively, are also preserved in the expected architecture. This confirms our assumption that identifying elements of consensus groups is valuable for improving the clustering results.

Furthermore, the union-based decomposition splits a non-negligible number of consensus groups. A large fraction of

TABLE II: Usefulness of Consensus-based Groups

Case Study	Bunch-NAHC				Spectral (k=GT)				Spectral (k=Bunch)			
	# consensus groups, $\hat{C}$	# (%) in Expected, $\hat{C}^P$	# split by Union, $\hat{C}^{S_U}$		# consensus groups, $\hat{C}$	# (%) in Expected, $\hat{C}^P$	# split by Union, $\hat{C}^{S_U}$		# consensus groups, $\hat{C}$	# (%) in Expected, $\hat{C}^P$	# split by Union, $\hat{C}^{S_U}$	
			total	in Expected			total	in Expected			total	in Expected
Bash	10	7 (70%)	5	5 (100%)	14	9 (64%)	4	2 (50%)	7	4 (57%)	1	1 (100%)
Hadoop	65	49 (75%)	7	4 (57%)	107	59 (55%)	21	9 (43%)	62	47 (76%)	14	14 (100%)
ArchStudio	73	65 (89%)	12	11 (92%)	105	73 (70%)	39	21 (54%)	90	75 (83%)	21	18 (86%)
Mozilla Firefox	301	281 (93%)	53	50 (94%)	170	118 (69%)	66	35 (53%)	322	291 (90%)	68	61 (90%)
X-wiki	410	364 (89%)	72	59 (82%)	363	87 (24%)	149	18 (12%)	659	308 (47%)	233	128 (55%)
PartsUnlimited	10	10 (100%)	0	0 (0%)	7	7 (100%)	1	1 (100%)	10	10 (100%)	2	2 (100%)
Average	144.8	129.3 (86%)	24.8	21.5 (70.8%)	127.7	58.3 (63.7%)	46.7	14.3 (52%)	191.7	122.5 (75.5%)	56.5	37.3 (88.5%)

these consensus groups (70.8%, 52%, and 88.5%, for the three different approaches, respectively) indeed appear in the expected architecture and thus need to be preserved. This suggests an opportunity for improving architectural recovery techniques by preserving agreements between decompositions produced by individual relationships.

To answer RQ1, the consensus-based strategy of combining information induced by decompositions performed by individual relationships appears useful. It has the potential to improve the quality of architectural recovery techniques that aim to consider multiple relationships simultaneously.

B. RQ2: Performance

To further investigate whether "locking together" elements of consensus groups (including those that are not preserved in the expected architectures) is worthwhile, we calculated the similarity between the obtained and expected decompositions. Table III shows the results for all considered similarity metrics, for each evaluated case study and approach. In particular, the "mega-rows" correspond to the subject applications we considered and, for each application, we further list the evaluated similarity metrics and the number of clusters produced by each decomposition. The columns correspond to different decomposition approaches we used: by individual relationship types (static and name-based similarity), using the union-based approach, and using the consensus-based approach.

The results show that, w.r.t. the *MoJoFM* metric, the consensus-based approach outperforms union in all but two cases: the PartsUnlimited project decomposed with BUNCH, where both approaches produce identical decompositions, and the Mozilla Firefox project decomposed with *Spectral (k=GT)*.

The decomposition produced by the consensus-based approach is not as good as the one produced by union for Mozilla Firefox because the majority of relevant consensus groups in this case, i.e., those present in the expected architecture, are relatively small, containing less than 10 elements each (3.6 elements on average). At the same time, there is an exceptional number of very large consensus groups, with 11 to 150 elements each (33.3 elements on average) that are not in the expected architecture. Keeping elements of these large consensus groups together comes at a price for the consensus-based approach, for all of the metrics. Consensus groups are also less accurate in this case as the expected architecture has largely unbalanced clusters (average size: 52.2, min: 1, max: 701, median: 22).

Interestingly, when running the *Spectral* clustering with a larger k, i.e., *k=Bunch*, the consensus groups become smaller and more precise: 90% of the 322 identified consensus groups are present in the expected architecture and these groups range between 2 and 7 in size (average 2.2).

The value of the *a2a* metric is higher for union- than for the consensus-based decomposition in three experiments: Bash with both *Spectral (k=GT)* and *(k=Bunch)* and Mozilla Firefox with *Spectral (k=Bunch)*. For Bash, the reason lies in duplicates: by keeping consensus groups together, the consensus-based approach ends up placing nine duplicated elements in three different clusters while both the union-based and the expected decompositions place these elements in two clusters only. As *a2a* is sensitive to duplicates, moving two of the elements from the "extra" cluster to both of their designated locations causes the consensus-based approach to have lower values here, albeit only slightly.

For Mozilla Firefox, the value of the metric is lower for the consensus-based approach because the metric assigns a higher weight to the cluster join operation, translating it into a set of moves. Consider, for example, the case where four elements from one expected cluster are split into two produced clusters: {a,b} and {c,d}. Unifying them takes one join operation but two moves. As the expected clusters are relatively unbalanced for Mozilla Firefox, per the discussion above, and as keeping consensus groups together typically results in more balanced clustering for *Spectral*, having to move many elements to larger clusters causes the value of *a2a* to be lower for the consensus-based decomposition.

Finally, the results of the $c2c_{cvg}$ metric show that, in the majority of cases, clusters produced by the consensus-based approach are more similar to the expected clusters than those produced by the union-based approach. An exception is, again, Bash, for all clustering approaches, as well as PartsUnlimited for *Spectral (k=Bunch)*, where the value of $c2c_{cvg}$ 10% metric is lower for the consensus-based than for union-based decomposition. Upon further inspection, as Bash has a relatively small number of relatively "dense" clusters, low-quality clusters can pass the 10% threshold of the $c2c_{cvg}$ 10% metric more easily, as they include just a few elements from at least one of the expected clusters. These clusters do not pass a higher similarity threshold though, as evident from the values of $c2c_{cvg}$ 33% and 50% metrics. That is, the union-based approach produces fewer clusters that are highly similar to the clusters in the expected decomposition in this case study as well, but the large number of low-similarity clusters, together with relatively large expected clusters, "inflate" the value of this metric. This is also the case for the decomposition produced by *Spectral (k=Bunch)* for PartsUnlimited.

TABLE III: Similarity Between Obtained and Expected Results for Different Decomposition Types

Case Study		Bunch-NAHC				Spectral (k=GT)				Spectral (k=Bunch)			
		Static	Name	Union	Consensus	Static	Name	Union	Consensus	Static	Name	Union	Consensus
Bash	MojoFM	58.14	32.43	48.29	**52.02**	54.49	22.97	63.46	**68.27**	59.8	24.32	48.08	**50**
	a2a	75.46	33.92	75.38	**75.87**	78.89	32.28	**83.99**	83.29	75.12	33.3	**75.67**	75.4
	c2c_cvg 10%	38.81	30	**31.96**	30	78.57	35.71	71.42	71.42	37.31	30	**33.33**	32.22
	c2c_cvg 33%	11.94	10	11.34	**12.22**	35.71	0	28.57	**42.86**	14.93	3.33	10	**12.22**
	c2c_cvg 50%	1.49	0	0	**2.22**	14.29	0	21.43	**35.71**	2.99	0	0	**1.11**
	# Clusters	67	30	97	90	14	14	14	14	67	30	90	90
Hadoop	MojoFM	46.04	37.82	42.18	**42.71**	52.16	36.26	39.02	**41.48**	46.04	36.84	42.36	**43.41**
	a2a	71.33	68.18	72.06	**72.31**	72.46	68.34	72.54	**72.64**	71.43	68.21	72.71	**72.78**
	c2c_cvg 10%	53.4	58.24	56.27	**56.31**	82.09	67.16	71.64	**73.13**	54.85	53.85	51.94	**53.88**
	c2c_cvg 33%	27.67	25.27	26.04	**27.18**	29.85	31.34	26.87	**28.35**	32.52	24.73	26.21	**27.67**
	c2c_cvg 50%	3.39	4.39	4.19	**4.37**	8.96	7.46	**8.96**	5.97	5.34	3.85	**5.83**	5.34
	# Clusters	206	182	215	206	67	67	67	67	206	182	206	206
Arch-Studio	MojoFM	58.45	55.62	58.44	**61.28**	73.02	57.12	61.63	**63.23**	61.33	53.18	59.33	**60.75**
	a2a	82.15	80.44	82.78	**82.97**	85.64	80.76	83.76	**83.96**	82.62	80.6	82.77	**83.24**
	c2c_cvg 10%	53.89	48.66	54.17	**54.19**	89.47	63.15	66.67	**70.18**	50.78	46.52	**53.63**	53.07
	c2c_cvg 33%	28.5	27.3	28.13	**29.05**	61.4	26.32	**35.09**	31.58	26.94	24.6	25.7	**27.93**
	c2c_cvg 50%	8.29	5.88	6.77	**7.82**	35.09	8.77	14.03	14.03	13.47	6.42	7.82	**10.61**
	# Clusters	193	187	193	179	57	57	57	57	193	187	179	179
Mozilla Firefox	MojoFM	72.41	64.07	68.65	**71.01**	77.41	68.23	**74.56**	68.41	72.5	55.13	63.96	**64.71**
	a2a	79.09	74.08	78.63	**79.41**	89.88	82.89	**89.03**	87.51	79.41	73.75	**79.52**	78.84
	c2c_cvg 10%	19.68	17.42	17.16	**19.44**	67.16	70.15	**77.61**	65.67	20.83	15.94	17.9	**18.74**
	c2c_cvg 33%	4.89	3.51	4.23	**5.03**	53.73	49.25	**56.72**	37.31	4.74	3.07	4.48	4.48
	c2c_cvg 50%	1.72	1.61	1.37	**1.81**	41.79	28.36	**35.82**	23.88	1.87	1.75	1.82	**2.1**
	# Clusters	696	684	804	715	67	67	67	67	696	684	715	715
X-wiki	MojoFM	52.54	48.53	52.07	**54.79**	50.59	33.13	31.61	**34.59**	58.3	41.86	52.81	**59.3**
	a2a	72.64	74.54	74.47	**74.6**	68.42	66.01	65.19	**65.81**	61	63.56	61.54	**62.36**
	c2c_cvg 10%	16.59	18.25	17.82	**22.3**	65.79	36.84	39.27	**55.26**	19.01	15.19	21.24	**21.64**
	c2c_cvg 33%	2.42	3.29	3.35	**3.69**	30.26	10.53	6.58	6.58	3.4	2.95	3.69	**5.54**
	c2c_cvg 50%	0.44	0.34	0.22	**0.66**	11.84	1.32	1.32	**2.63**	0.76	0.79	1.45	**2.11**
	# Clusters	910	882	926	758	76	76	76	76	910	882	758	758
Parts-Unlimited	MojoFM	63.33	90.32	71.43	71.43	76.67	54.84	82.86	82.86	56.67	90.32	71.43	**77.14**
	a2a	58.12	63.91	63.79	63.79	63.27	59.21	69.49	**69.91**	57.26	63.91	63.37	**66.26**
	c2c_cvg 10%	84.62	100	83.33	83.33	100	100	100	100	92.31	100	**91.67**	75
	c2c_cvg 33%	15.38	71.43	16.67	16.67	100	60	100	100	15.38	71.43	16.67	**33.33**
	c2c_cvg 50%	0	14.3	8.33	8.33	0	20	60	**80**	0	14.3	8.33	8.33
	# Clusters	13	7	12	12	5	5	5	5	13	7	12	12

For ArchStudio and Hadoop, with $c2c_{cvg}$ 33% and 50%, respectively: both these applications have a very large number of small clusters in the expected decomposition (the median number of elements is 3 and 4, respectively). Thus, union-based decompositions for the *Spectral* approach, which tend to generate many clusters of size 1, have an advantage here.

We also note that the quality of decompositions produced by *Spectral (k=GT)* are higher than the other approaches for the majority of strategies, case studies, and metrics. We believe it is largely because decompositions produced by this configuration contain the same number of clusters as the expected architecture while the other configurations produce decompositions with vastly more clusters.

To answer RQ2.1, the consensus-based approach consistently produces decompositions that are more similar to the expected architecture than the union-based approach. For BUNCH, all but one metric for one of the case studies is higher for consensus-based decompositions. For *Spectral*, *MoJoFM* prefers consensus-based decompositions as they induce more balanced clusters that are easier to join. The results are mixed w.r.t. metrics that give higher weight to the join operations, such as *a2a* and $c2c_{cvg}$.

For the runtime of the approaches, *Spectral* finishes in a matter of a few minutes for all applications and strategies. The increased benefits in accuracy of the consensus-based strategy with BUNCH comes at the price of increased execution time because our consensus-based approach relies on decomposing the applications by individual relationships first, as shown in Figure 2. Yet, the execution time of the consensus-based decomposition itself is, in fact, shorter than that of the union-based because certain moves are prevented by "locking" consensus group elements together. For example, it took 2 hours and 20 minutes for the union-based approach to decompose the Bash application while the consensus-based decomposition took one hour only. It is prefaced by 2 hours and 32 minutes total time taken to perform decompositions by static and name-based similarity relationships. Overall, the execution time of both consensus and union approaches for BUNCH ranges between several minutes (PartsUnlimited), several hours (Bash), several days (Hadoop and ArchStudio), and several weeks (X-wiki and Mozilla Firefox).

To answer RQ2.2, all decompositions that use spectral clustering terminate within several minutes. For BUNCH, consensus-based strategy increases the overall execution time compared with the union-based strategy by 55.6%, on average.

C. Lessons Learned and Possible Next Steps

Our analysis of case studies and decomposition results lead to the following observations and suggestions for future research:

1. While the general-purpose clustering technique that we used in our analysis was extremely efficient w.r.t. execution time, selecting a proper number of desired clusters can largely affect the quality of the produced results. Future work could look at more appropriate general-purpose clustering techniques

and map their properties w.r.t. the architectural recovery task. Exploring ways to pipeline approaches, e.g., to start from a spectral decomposition as the initial population for BUNCH, could be another subject of possible future work.

2. When focusing on architectural-recovery-specific clustering, we observed that the number of clusters in decompositions produced by BUNCH is always an order of magnitude higher than in the expected results. As merging distinct clusters could be considered a simpler operation than finding and organizing related elements (as also reflected in the *MoJoFM* metric), we believe this design decision is reasonable. Yet, future work could focus on approaches that produce less fine-grained decompositions, as decompositions with hundreds of clusters could be difficult to handle in practice.

3. We also observed that, for BUNCH, the consensus-based approach consistently produces a smaller number of clusters than the union-based approach, resulting in a number of clusters closer to the number of clusters in the expected decomposition. Moreover, consensus groups induce more reliable clustering results, albeit at the expense of longer execution time. Future work could focus on how to perform decomposition by multiple types of relationships simultaneously, thus decreasing execution time while keeping the benefits of consensus. Such work could draw inspiration from consensus-based clustering approaches studied in other areas [16], [31], [51], [67], [68].

4. Our current evaluation is limited to considering only two relationships at a time. Further work is needed to evaluate the usefulness of the consensus-based approach for more than two types of relationship. Specifically, identifying "agreements" between a larger number of relationships will inevitable result in smaller, yet, more reliable consensus groups. Additional experiments are needed to assess whether these groups are of a reasonable size and whether they would be preserved by the union-based decomposition in any case. While our current experience indicates that union-based decompositions often split consensus elements apart, future work is needed to investigate the properties of clustering approaches that cause such behavior.

5. Finally, the quality and usefulness of decompositions is highly sensitive to the evaluation metric used. Specifically, consensus-based decompositions outperformed union-based w.r.t. the *MoJoFM* metric in the majority of cases but did not perform as well for the *a2a* and $c2c_{cvg}$ metrics for the spectral clustering. We believe it is important to adjust the approach to the needs of the architect performing the decomposition.

VI. Limitations and Threats to Validity

For **external validity**, our results may not generalize beyond our selected case study applications. We attempted to mitigate this threat by selecting a variety of popular open-source applications with different functionalities, sizes, and programming paradigms. We thus believe our selection of applications is reliable.

Another important factor that may affect the validity of our results is the accuracy of the expected decomposition used for our case studies. We mitigated this threat by borrowing case studies from existing literature, where knowledgeable developers and architects have aided in the production of the expected architectures. Yet, as there does not necessarily exist a unique, correct architecture for a system [21], [34], our results might not generalize for target architectures that reflect different decompositional goals.

Finally, our results could also be affected by the selection of clustering algorithms we experiment with. We attempted to mitigate this threat by using popular third-party clustering approaches, both from the architectural recovery community and the general-purpose machine learning field.

For **internal validity**, deficiencies of the underlying tools our approach uses, such as the static analysis techniques we utilized in our work, might have affected the accuracy of the results. We controlled for this threat by manually analyzing all the considered cases and confirming their correctness.

VII. Related work

A number of authors studied the quality of architectural recovery techniques and, specifically, the use of relationships in architectural recovery. For example, Abreu and Goulao [22], Bavota et al. [19], as well as Candela et al. [23] investigated how class coupling, as captured by structural, dynamic, semantic, and logical coupling measures, aligns with developers' perception of coupling. Similarly, Lutellier et al. [43], [44] explored the impact of relationships on architectural recovery approaches, focusing on structural relationships only. Garcia et al. [33] performed a comprehensive analysis of software architecture recovery techniques showing a relatively low accuracy for most of the analyzed approaches. None of these works focused on exploring approaches for combining multiple relationships, as we do in our study.

The clustering ensemble concepts from the field of machine learning and data science are also related to our work [16], [31], [51], [67], [68]. The main idea in these techniques is to combine multiple partitionings of a set of objects into a single consolidated clustering without accessing the features or algorithms that determined these partitionings, which can be translated into our problem of combining decompositions performed by multiple individual relationship types. While these techniques were previously applied in multiple domains, such as social network clustering, exploring their applicability for architectural recover is a subject of possible future work.

VIII. Conclusion

In this paper, we defined and explored different strategies for combining multiple relationship types during architectural recovery and microservice extraction processes. We introduced a new consensus-based strategy for combining relationships and empirically demonstrated that it outperforms the union-based strategy in the majority of cases. We discussed the advantages and disadvantages of each strategy and outlined directions for possible future work.

Acknowledgments: This work has been partially supported by IBM Canada.

REFERENCES

[1] "Apache Hadoop," https://hadoop.apache.org/.
[2] "ArchStudio 4," https://github.com/isr-uci-edu/ArchStudio4.
[3] "Bash v4.2," https://github.com/bminor/bash/releases/tag/bash-4.2.
[4] "Comparing Software Architecture Recovery Techniques Using Accurate Dependencies – Ground Truth Architectures," https://www.cs.purdue.edu/homes/lintan/ArchRecovery/.
[5] "Most common english words," https://www.textfixer.com/tutorials/common-english-words.txt.
[6] "Mozilla Firefox Developer Preview, version 1.9.3a4," https://ftp.mozilla.org/pub/firefox/releases/devpreview/1.9.3a4/.
[7] "Natural Language Toolkit," https://www.nltk.org/.
[8] "Parts Unlimited MRP," https://github.com/microsoft/PartsUnlimitedMRP.
[9] "Parts Unlimited MRP Microservices," https://github.com/microsoft/partsunlimitedMRPmicro.
[10] "Powered by Apache Hadoop," https://cwiki.apache.org/confluence/display/HADOOP2/PoweredBy.
[11] "Understand, SciTools," https://scitools.com, [Online; accessed on 2022].
[12] "XWiki Platform," https://github.com/xwiki/xwiki-platform.
[13] M. Abdullah, W. Iqbal, and A. Erradi, "Unsupervised Learning Approach for Web Application Auto-Decomposition into Microservices," *Journal of Systems and Software (JSS)*, vol. 151, pp. 243–257, 2019.
[14] J. Ahn, E. Boerstra, and J. Rubin. (2022) Supplementary Materials. https://resess.github.io/artifacts/CombiningRelationships/.
[15] O. Al-Debagy and P. Martinek, "Dependencies-based Microservices Decomposition Method," *International Journal of Computers and Applications*, pp. 1–8, 2021.
[16] T. Alqurashi and W. Wang, "A New Consensus Function Based on Dual-similarity Measurements for Clustering Ensemble," in *International Conference on Data Science and Advanced Analytics (DSAA)*, 2015, pp. 1–9.
[17] P. Andritsos and V. Tzerpos, "Information-theoretic software clustering," *IEEE Transactions on Software Engineering*, vol. 31, no. 2, pp. 150–165, 2005.
[18] L. Baresi, M. Garriga, and A. De Renzis, "Microservices Identification Through Interface Analysis," in *European Conference on Service-Oriented and Cloud Computing (ESOCC 2017)*, 2017, pp. 19–33.
[19] G. Bavota, B. Dit, R. Oliveto, M. D. Penta, D. Poshyvanyk, and A. D. Lucia, "An Empirical Study on the Developers' Perception of Software Coupling," in *International Conference on Software Engineering (ICSE)*, 2013, pp. 692–701.
[20] F. Beck and S. Diehl, "Visual comparison of software architectures," *Information Visualization*, vol. 12, no. 2, pp. 178–199, 2013.
[21] I. Bowman, R. Holt, and N. Brewster, "Linux as a Case Study: Its Extracted Software Architecture," in *International Conference on Software Engineering*, 1999, pp. 555–563.
[22] F. Brito e Abreu and M. Goulao, "Coupling and Cohesion as Modularization Drivers: Are We Being Over-Persuaded?" in *European Conference on Software Maintenance and Reengineering (CSMR)*, 2001, pp. 47–57.
[23] I. Candela, G. Bavota, B. Russo, and R. Oliveto, "Using Cohesion and Coupling for Software Remodularization: Is It Enough?" *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 25, no. 3, 2016.
[24] L. Carvalho, A. Garcia, T. E. Colanzi, W. K. G. Assunção, J. A. Pereira, B. Fonseca, M. Ribeiro, M. J. de Lima, and C. Lucena, "On the Performance and Adoption of Search-Based Microservice Identification with toMicroservices," in *IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2020, pp. 569–580.
[25] M. Chatterjee, S. K. Das, and D. Turgut, "WCA: A Weighted Clustering Algorithm for Mobile Ad Hoc Networks," *Cluster Computing*, vol. 5, no. 2, pp. 193–204, 2002.
[26] F. R. Chung, "Spectral Graph Theory," *CBMS, American Math Society*, 1997.
[27] A. Corazza, S. Di Martino, V. Maggio, and G. Scanniello, "Investigating the Use of Lexical Information for Software System Clustering," in *European Conference on Software Maintenance and Reengineering (CSMR)*, 2011, pp. 35–44.
[28] D. Escobar, D. Cárdenas, R. Amarillo, E. Castro, K. Garcés, C. Parra, and R. Casallas, "Towards the Understanding and Evolution of Monolithic Applications as Microservices," in *XLII Latin American Computing Conference (CLEI)*, 2016, pp. 1–11.
[29] S. Eski and F. Buzluca, "An Automatic Extraction Approach: Transition to Microservices Architecture from Monolithic Application," in *Proceedings of the 19th International Conference on Agile Software Development (XP'18): Companion*, 2018, pp. 1–6.
[30] B. S. Everitt, S. Landau, and M. Leese, *Cluster Analysis*, 4th ed. Wiley Publishing, 2009.
[31] A. L. N. Fred and A. K. Jain, "Combining Multiple Clusterings Using Evidence Accumulation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, no. 6, pp. 835–850, 2005.
[32] J. Garcia, D. Popescu, C. Mattmann, N. Medvidovic, and Yuanfang Cai, "Enhancing Architectural Recovery Using Concerns," in *IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2011, pp. 552–555.
[33] J. Garcia, I. Ivkovic, and N. Medvidovic, "A Comparative Analysis of Software Architecture Recovery Techniques," in *International Conference on Automated Software Engineering (ASE)*, 2013, pp. 486–496.
[34] J. Garcia, I. Krka, C. Mattmann, and N. Medvidovic, "Obtaining Ground-Truth Software Architectures," in *International Conference on Software Engineering (ICSE)*, 2013, pp. 901–910.
[35] J. A. Hartigan and M. A. Wong, "A K-Means Clustering Algorithm," *Journal of the Royal Statistical Society*, vol. 28, pp. 100–108, 1979.
[36] A. Jain, K. Nandakumar, and A. Ross, "Score Normalization in Multimodal Biometric Systems," *Pattern Recognition*, vol. 38, no. 12, pp. 2270–2285, 2005.
[37] W. Jin, T. Liu, Y. Cai, R. Kazman, R. Mo, and Q. Zheng, "Service Candidate Identification from Monolithic Systems based on Execution Traces," *IEEE Transactions on Software Engineering (TSE)*, 2019.
[38] A. K. Kalia, J. Xiao, C. Lin, S. Sinha, J. J. Rofrano, M. Vukovic, and D. Banerjee, "Mono2Micro: An AI-based Toolchain for Evolving Monolithic Enterprise Applications to a Microservice Architecture," in *Tool Demos of ESEC/FSE*, 2020, pp. 1606–1610.
[39] L. J. Kirby, E. Boerstra, Z. J. Anderson, and J. Rubin, "Weighing the Evidence: On Relationship Types in Microservice Extraction," in *International Conference on Program Comprehension (ICPC)*, 2021, pp. 358–368.
[40] D. M. Le, P. Behnamghader, J. Garcia, D. Link, A. Shahbazian, and N. Medvidovic, "An Empirical Study of Architectural Change in Open-Source Software Systems," in *Conference on Mining Software Repositories*, 2015, pp. 235–245.
[41] K.-S. Lee and C.-G. Lee, "Identifying Semantic Outliers of Source Code Artifacts and Their Application to Software Architecture Recovery," *IEEE Access*, vol. 8, pp. 212 467–212 477, 2020.
[42] J. Lewis and M. Fowler, "Microservices: a Definition of This New Architectural Term," https://www.martinfowler.com/articles/microservices.html, 2014.
[43] T. Lutellier, D. Chollak, J. Garcia, L. Tan, D. Rayside, N. Medvidovic, and R. Kroeger, "Comparing Software Architecture Recovery Techniques Using Accurate Dependencies," in *International Conference on Software Engineering (ICSE)*, 2015, pp. 69–78.
[44] ——, "Measuring the Impact of Code Dependencies on Software Architecture Recovery Techniques," *IEEE Transactions on Software Engineering*, vol. 44, pp. 159–181, 2017.
[45] S. Mancoridis, B. Mitchell, Y.-F. Chen, and E. Gansner, "Bunch: A clustering tool for the recovery and maintenance of software system structures," 02 1999, pp. 50 – 59.
[46] O. Maqbool and H. Babri, "Hierarchical Clustering for Software Architecture Recovery," *IEEE Transactions on Software Engineering*, vol. 33, no. 11, pp. 759–780, 2007.
[47] T. Matias, F. F. Correia, J. Fritzsch, J. Bogner, H. S. Ferreira, and A. Restivo, "Determining Microservice Boundaries: A Case Study Using Static and Dynamic Software Analysis," in *Software Architecture*, 2020, pp. 315–332.
[48] G. Mazlami, J. Cito, and P. Leitner, "Extraction of Microservices from Monolithic Software Architectures," in *International Conference on Web Services (ICWS)*, pp. 524–531.
[49] J. Misra, K. Annervaz, V. Kaulgud, S. Sengupta, and G. Titus, "Software Clustering: Unifying Syntactic and Semantic Features," in *Working Conference on Reverse Engineering*, 2012, pp. 113–122.
[50] B. S. Mitchell and S. Mancoridis, "On the automatic modularization of software systems using the Bunch tool," *IEEE Transactions on Software Engineering*, vol. 32, no. 3, pp. 193–208, March 2006.
[51] S. Monti, P. Tamayo, J. P. Mesirov, and T. R. Golub, "Consensus Clustering: A Resampling-Based Method for Class Discovery and Visualization of Gene Expression Microarray Data," *Machine Learning*, vol. 52, no. 1-2, pp. 91–118, 2003.

[52] O. Mustafa, J. M. Gómez, M. Hamed, and H. Pargmann, "GranMicro: A black-box based approach for optimizing microservices based applications," in *From Science to Society*, 2018, pp. 283–294.

[53] A. Y. Ng, M. I. Jordan, and Y. Weiss, "On Spectral Clustering: Analysis and an Algorithm," in *International Conference on Neural Information Processing Systems: Natural and Synthetic*, 2001, p. 849–856.

[54] C. Patel, A. Hamou-Lhadj, and J. Rilling, "Software Clustering Using Dynamic Analysis and Static Dependencies," in *European Conference on Software Maintenance and Reengineering (CSMR)*, 2009, pp. 27–36.

[55] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. VanderPlas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[56] I. Pigazzini, F. A. Fontana, and A. Maggioni, "Tool Support for the Migration to Microservice Architecture: An Industrial Case Study," in *Software Architecture*, 2019, pp. 247–263.

[57] A. Rathee and J. Chhabra, *Software Remodularization by Estimating Structural and Conceptual Relations Among Classes and Using Hierarchical Clustering*, 2017, pp. 94–106.

[58] Z. Ren, W. Wang, G. Wu, C. Gao, W. Chen, J. Wei, and T. Huang, "Migrating Web Applications from Monolithic Structure to Microservices Architecture," in *Asia-Pacific Symposium on Internetware (Internetware)*, 2018, pp. 1–10.

[59] M. Schmitt Laser, N. Medvidovic, D. M. Le, and J. Garcia, "ARCADE: An Extensible Workbench for Architecture Recovery, Change, and Decay Evaluation," in *European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, p. 1546–1550.

[60] scikit learn, "User Guide: Clustering," https://scikit-learn.org/stable/modules/clustering.html.

[61] A. Selmadji, A.-D. Seriai, H. L. Bouziane, C. Dony, and R. O. Mahamane, "Re-architecting oo software into microservices," in *European Conference on Service-Oriented and Cloud Computing (ESOCC)*, 2018, pp. 65–73.

[62] A. Shokoufandeh, S. Mancoridis, T. Denton, and M. Maycock, "Spectral and Meta-heuristic Algorithms for Software Clustering," *Journal of Systems and Software*, vol. 77, no. 3, pp. 213–223, 2005.

[63] H. Sözer, "Evaluating the Effectiveness of Multi-level Greedy Modularity Clustering for Software Architecture Recovery," in *Software Architecture*, 2019, pp. 71–87.

[64] StackOverflow, "Is there a way to enforce that a set of points are assigned to the same class when clustering in sklearn or other clustering library?" https://stackoverflow.com/questions/67151897/is-there-a-way-to-enforce-that-a-set-of-points-are-assigned-to-the-same-class-wh.

[65] I. Stavropoulou, M. Grigoriou, and K. Kontogiannis, "Case Study on Which Relations to Use for Clustering-Based Software Architecture Recovery," *Empirical Software Engineering*, vol. 22, no. 4, p. 1717–1762, 2017.

[66] H. Streekmann, *Clustering-Based Support for Software Architecture Restructuring*. Wiesbaden, Germany: Springer Vieweg Verlag, 2011.

[67] A. Strehl and J. Ghosh, "Cluster Ensembles – A Knowledge Reuse Framework for Combining Multiple Partitions," *Journal of Machine Learning Research*, vol. 3, pp. 583–617, 2002.

[68] S. Swift, A. Tucker, V. Vinciotti, M. Nigel, C. Orengo, X. Liu, and P. Kellam, "Consensus Clustering and Functional Interpretation of Gene-expression Data," *Genome Biology*, vol. 5, no. 11, p. R94, 2004.

[69] D. Taibi and K. Systä, "From Monolithic Systems to Microservices: a Decomposition Framework Based on Process Mining," in *8th International Conference on Cloud Computing and Services Science (CLOSER)*, 2019.

[70] N. Teymourian, H. Izadkhah, and A. Isazadeh, "A fast clustering algorithm for modularization of large-scale software systems," *IEEE Transactions on Software Engineering*, 2020.

[71] V. Tzerpos and R. Holt, "ACDC: An Algorithm for Comprehension-Driven Clustering," 02 2000, pp. 258–267.

[72] S. Wu, *Score Normalization*, 2012, pp. 19–42.

[73] Zhihua Wen and V. Tzerpos, "An Effectiveness Measure for Software Clustering Algorithms," in *IEEE International Workshop on Program Comprehension (WPC)*, 2004, pp. 194–203.