UNIVERSITY OF GOTHENBURG

On The Significance of Relationship Directions in Clustering Algorithms for Reverse Engineering

Bachelor of Science Thesis in the Software Engineering and Management Programme

David Jensen
Andreas Lundkvist

University of Gothenburg
Chalmers University of Technology
Department of Computer Science and Engineering
Göteborg, Sweden, June 2016

On The Significance of Relationship Directions in Clustering Algorithms for Reverse Engineering

David Jensen
Andreas Lundkvist

Examiner: Regina Hebig

University of Gothenburg
Chalmers University of Technology
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden June 20

On The Significance of Relationship Directions in Clustering Algorithms for Reverse Engineering

David Jensen and Andreas Lundkvist
Department of Computer Science and Engineering
University of Gothenburg
Gothenburg, Sweden
david_jensen3@hotmail.com, gu@galdiuz.com

***Abstract*—Software clustering is a common technique applied to simplify reverse engineered software models. These algorithms commonly classify similarity between nodes based on their relationships. However little research exists that discusses the importance of the direction of these relationships.**

In this paper we provide empirical data for how treating direction in entity relationships affect the recovery accuracy of hierarchical clustering algorithms. We test variations of a hierarchical clustering algorithm on several open-source systems and compare their results, and conclude that relationship direction does not have an significant impact on recovery accuracy. As such, researchers may opt to implement hierarchical clustering algorithms using only one direction of relations instead of both, and still get similar results for less computational cost.

I. Introduction

Hierarchical clustering algorithms in reverse engineering commonly define features of nodes as their relationships to other nodes. This idea originates from Schwanke and Platoff's paper "Cross references are features" [1]. Schwanke and Platoff also propose distinguishing between user-names (features representing other entities that use the entity) from names-used (features representing what the entity is using). Neither whether both of these two variants of relationship features are taken into account, nor whether they are distinguished, is something later researches are explicit about, and no rationale is given for why they are or are not. Likewise, Schwanke and Platoff's proposal contains no justification to why this separation should be done. More specifically, they do not provide any empirical data comparing the differences between including or not including both features as well as separating or combining them.

Hence we will examine the different outcomes of including user-names and names-used relationships, as well as classifying them differently versus ignoring their direction. We present the following hypotheses:

- $H^1{}_0$ - Including user-names in addition to names-used does not increase the recovery accuracy of hierarchical clustering algorithms
- $H^1{}_1$ - Including user-names in addition to names-used increases the recovery accuracy of hierarchical clustering algorithms.
- $H^2{}_0$ - Distinguishing between names-used and user-names when calculating cluster similarity does not increase the recovery accuracy of hierarchical clustering algorithms.
- $H^2{}_1$ - Distinguishing between names-used and user-names when calculating cluster similarity increases the recovery accuracy of hierarchical clustering algorithms.

We will test these hypotheses by implementing variations of a hierarchical clustering algorithm where relationship direction is handled differently and compare their results. The tests will be done on several open-source systems of various domains and size. This data can then be used to find how directions should be treated in order to get the highest accuracy in hierarchical clustering algorithms.

The rest of the paper is structured as follows. Section II contains definitions of concepts used in the paper. Section III contains motivation behind the research, while section IV contains background information and related research. Section V explains our methodology, section VI contains the results of the tests, and in section VII we discuss the results. In section VIII we discuss the limits of our research and recommendations for further research, and finally, we give our conclusion in section IX.

II. Definitions

The following list contains definitions of concepts used in this paper.

- *Reverse engineering* - Recreating or extracting information from a system, e.g. the software architecture [2]. In this paper we use reverse engineering and recovery interchangeably.
- *Cluster* - A grouping of entities, in our case classes.
- *Hierarchical clustering* - A method of building binary trees of clusters, paired together based on cluster similarity. Hierarchical clustering algorithms are generally divided into two types; agglomerative, where clusters are paired bottom up, and divisive, where one large cluster is split top down [3].
- *Feature vector* - A vector defining the features of an entity (e.g. class) that are used when comparing the entity to others [4].
- *Dependency* - A dependency occurs from component A to component B when component A depends on component B by for example calling one of component B's methods. Denoted in graphs by an arrow from A to B (A→B, cf. fig 1). During data extraction we regard a class X as being

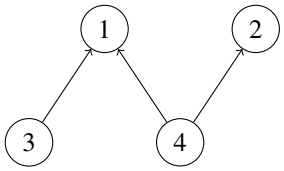

Fig. 1. A relation graph for four classes (1, 2, 3, and 4). Arrows denotes dependencies, e.g. 3 is dependent on 1.

dependent on Y if the typename for Y occurs anywhere inside the declaration or definition of X.

- *Names-used* - The entities an entity is using, i.e. which entities it has a dependency to [1].
- *User-names* - The entities that are using an entity, i.e. which entities that have a dependency to it [1].
- *Weighted Combined Algorithm (WCA)* - A hierarchical clustering algorithm created by Maqbool and Babri [4] which is the most successful hierarchical algorithm. We will use this algorithm for our tests.
- *MojoFM* - An algorithm that measures how similar two sets of clusters are, expressed as a percentage [5]. We will use MojoFM to compare our algorithm implementations.

III. Motivation

Schwanke and Platoff's paper gives no motivation for why both feature kinds should be including, nor why they should be distinguished between [1]. We will in the following section try to provide motivation for doing so.

A. Including user-names in the feature vector

Many cluster algorithms in information retrieval use the concept of feature vectors. A feature vector denotes the properties of an object that can then be used to classify an object and its similarity towards other objects. In the context of clustering software components with the sibling link approach, the feature vector denotes the relations a component have in a dependency graph. For example, when two components X and Y both depend on component Z, they are seen as sharing the feature Z. Using the words of Schwanke and Platoff, X and Y can be referred to as users or "user-names" and Z as "names-used" [1]. Although they give no rationale to why one should include the names of users in the feature vector, it is simple to present a case where the final result is improved by doing so. The rationale we give for our first hypothesis is that including user-names features causes components to be similar even when there is an absence of a shared dependency, if there exists a shared context from which they are used. For example, figure 1 denotes a graph of four components. In this graph, 3 and 4 would be deemed similar if "names-used" are included in the usage vector because of their shared dependency 1, whereas 1 and 2 would only be deemed similar if "user-names" are included because they are both used by 4.

B. Separating names-used and user-names in the feature vector

Schwanke and Platoff make a clear distinction between the relationships components have to each other and they distinguish "user-names" features from "names-used" by tagging user-names features with # and names-used with & [1]. Yet they do not provide the rationale for this separation (we will attempt to in the next paragraph) and it is often very unclear whether researchers follow this idea of distinguishing names-used and user-names. For example, in their paper Maqbool and Babri mention only that "edges represent the features of the nodes they connect, and similarity is measured by looking at features that two nodes have in common" [4, p.2], while they further down in their paper mention what kinds of dependencies they count as features (routines, global variables, and user types) [4].

We will now discuss the impact of not distinguishing between names-used and user-names features and give rationale for distinguishing them. One can ignore the directions of component relationships by looking only at them as connectors between entities that either exists or not. However, this means components that use a component Z will be deemed similar to components being used by component Z. We would argue that classifying relationships differently depending on whether they imply using or being used copes more accurately with the way we naturally layer components. For example, fig. 2 depicts a system of three layers, A, B, and C, where every class in A are used by classes in B, which in turn are used by classes in C. This follows a common philosophical idea in layered design which is that there exists a constant direction (either up or down) through which dependency flows [6], [7]. Following this philosophy, it does not make sense to ignore the direction of the relationships classes in A and C has to those in B. This would merge A and C's classes creating only two distinct clusters. Furthermore these two clusters B and AC would then indirectly create a circular reference on the subsystem level, something which is usually avoided within software design [8].

IV. Background

Documentation covering system designs often fall behind the present state of systems and may sometimes be lacking in detail or not exist at all. A fast way to recover missing documentation is to rely on reverse engineering [9]. However, reverse engineered models have a tendency to become overly complex since they do not contain the abstraction mechanisms usually applied manually by architects [10]. Some of these abstractions can be achieved by methods of filtering software entities. Another important simplification mechanism is reorganizing software into subsystems. Using clustering algorithms, reverse engineered software models can be re-modularized into new subsystems. These new subsystems can be used as models to help understand a legacy system or be used as proposals for re-factoring and criticize current architectures.

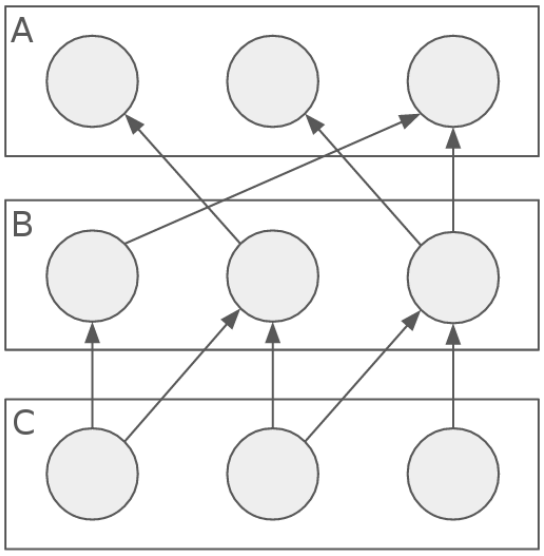

Fig. 2. Relationships between 3 subsystems

A. Clustering algorithms

The most successful clustering algorithms can be characterized as those relying on developing similarity functions [11] or those using structural discovery together with a quality metric. Examples of the latter are Bunch [12], "Architectural Recovery using Concerns" (ARC) [13], and "Algorithm for Comprehension-Driven Clustering" (ACDC) [14], three clustering algorithms that do not use similarity functions. However, as shown by Garcia et al. [15] even the most promising methods based on similarity functions, such as Maqbool and Babri's Weighted Combined Algorithm [4], are outperformed by both ACDC and ARC. Out of the 8 systems tested by Garcia et al., ARC and ACDC produced an average accuracy of 58.76% and 55.94% respectively, whereas WCA attained an average of 43.58% accuracy.

B. Hierarchical agglomerative clustering

When it comes to clustering algorithms using similarity functions, they are most commonly based on a hierarchical agglomerative clustering algorithm which in turn is the most popular algorithms used in design recovery [16]. The most successful hierarchical agglomerative algorithms are the Scalable Information Bottleneck (LIMBO) [17] and WCA [18].

Hierarchical agglomerative clustering algorithms creates clusters bottom up by continuously merging the two most similar clusters. In software clustering similarity is defined as a function over two clusters' feature vectors where the feature vectors describe the clusters' relationships to other clusters.

C. Linkage methods

When a non-singleton cluster is formed it should be able to be contained in another cluster. For this to happen one needs to compare the similarity between non-singleton clusters and other clusters. A linking method is therefore used to transfer similarity measures from child clusters into the newly formed parent cluster.

Maqbool and Babri proposed a new algorithm for finding the similarity of a newly formed cluster with another cluster, or rather how to define the feature vector of a non-singleton cluster [4]. Previous methods simply used the union or intersection of the two feature vectors. The combined weighted linkage however, keeps track of how many nodes out of the total nodes in a cluster has said feature. When calculating association coefficients the sum of the intersection of features appearing in both clusters is then weighted by the number of nodes in each cluster that share a certain feature in relation to the cluster's size.

Because the cluster size is taken into account the features become non-binary instead of binary, i.e. instead of a component having a feature or not there is a degree to how important the feature is. Furthermore, Maqbool and Babri propose a new similarity metric for non-binary features called the unbiased-Ellenberg:

$$E_u = \frac{1/2 * Ma}{1/2 * Ma + b + c}$$

"Here Ma represents the sum of features that are common in both entities, and b and c represent the count of features that are present in one entity and not in the other" [4, p.6]. Their unbiased-Ellenberg is based on the Ellenberg metric, which in turn is the non-binary counterpart of the Jaccard coefficient that has been proven to produce good results [19].

D. Accuracy versus comprehension

We believe that most subsystems that exist in as-is architectures to some extent serve a purpose for comprehensibility (although it may not be their main purpose). Likewise, most efforts that have been done to increase comprehension has proven efficient in recovery accuracy comparisons [15].

There is also the fact that what defines a meaningful or comprehensible subsystem is vague and varies between domains and among different people, which is partly a reason for the low accuracy rating of the current state-of-the-art clustering algorithms. This is not an argument for not applying comprehension research, but just a mention that it requires consulting with domain experts, something that was out of scope for our research. Finally, there is also the fact that there seem to exist a lot of academic interest in clustering for recovery accuracy [4], [5], [15], [13], [16] as well as a big body of knowledge for which we want to increase understanding, investigate methods, and increase replicateability.

V. Methodology

In order to test our hypotheses we will run four sets of benchmarks with different variants of WCA on the data sources. We compare the different variants capability of recreating projects by clustering classes.

The first set will be using both user-names and names-used combined, the second classifies user-names separately

TABLE I
CHOSEN REPOSITORIES

Repository	Domain	Lines of Code	Stars ★
AutoMapper/AutoMapper	Data/Cloning	35170	3494
MiniProfiler/dotnet	Profiling api	19812	625
DotNetOpenAuth/DotNetOpenAuth	Authorization api	182114	1153
aspnet/EntityFramework	Object relational mapper	257452	3285
schambers/fluentmigrator	Database migration	61075	1190
JeremySkinner/FluentValidation	Business objects validator	21075	1368
Mono/MonoGame	Game development kit	187784	3478
NancyFX/Nancy	Web framework	87145	4071
octokit/octokit.net	Github client library	76813	942
OpenRA/OpenRA	Game engine	105664	3018
ShareX/ShareX	Screenshot application	142486	3620
SignalR/SignalR	Web framework	49904	5713
hbons/SparkleShare	File server	13836	3616
Wox-launcher/Wox	Application launcher	12978	2854

★ As of 2016/05/24

from names-used, the third set uses only names-used, while the fourth uses only user-names. These benchmarks will be compared to the ground-truth data using a cluster recovery accuracy metric. We will be using the unbiased Ellenberg measure for cluster similarity, since it has been proven to provide the best results [4], [15]. The results of the first and second benchmark will be compared to the third and fourth to answer our first hypothesis, while the second hypothesis will be answered by comparing the results of the first and second benchmark.

The implementation of our algorithms, the data extractor and the benchmark tests can be found at: https://github.com/davidkron/Clustering.

A. Data sources

The algorithm was tested on open-source systems from GitHub. The repositories were chosen by sorting all repositories with language C# on stars (a popularity measurement) using GitHub's advanced search. From the top 50 we extracted 15 repositories, with the goal to maximize the diversity of domains and sizes of the sample.

After the original set of repositories was selected we removed all those not fulfilling our inclusion criteria, which were that they needed to 1) be primarily a C# system and 2) include at least two projects in the solution. This resulted in us having to remove shadowsocks/shadowsocks-windows (since it contained only one project) from our data set, resulting in the list of repositories you can find in table I. We have published the extracted data at https://github.com/davidkron/parsed-csharp-repos.

B. Data extraction

We used the .NET compiler platform Roslyn to extract an abstract syntax tree and class dependencies from the projects. The projects represent the ground truth data. From the ground truth data we extract all classes and their dependencies, discarding other information such as which project they belong to. These classes and their dependencies are used as input to the clustering algorithm.

C. Validation

We use MojoFM, a clustering recovery metric, to compare the results from the different algorithm variants with the ground truth data [5]. MojoFM has been presented as preferable over the precision-recall metric commonly used in information retrieval and pattern recognition because it is less influenced by the size of the clusters [20]. MojoFM compares two sets of clusters and tells how similar they are to each other by calculating the amount of move and join operations that would be required to turn one into the other. The similarity is presented in a value between 1 and 0, with a value of 1 (or 100%) being identical and a value of 0 being no similarities at all. This allows us to assess the effectiveness of the algorithms by comparing the architecture generated by the algorithm to the original architecture.

For every benchmark, the result of the cluster algorithm is cut using a static cutting algorithm to gain flat clusters (clusters not containing other clusters). This is because MojoFM requires flat clusters, and since WCA outputs hierarchical clusters the results need to be flattened before it can be measured. These flat clusters are then compared to the ground-truth clusters using MojoFM. As mentioned above, this will gives numbers on how accurate the different benchmarks are, and allows us to compare them to each other. We will test whether the results are statistically significant using a paired t-test, a test that assesses how statistically different the means of two groups are. We use a paired t-test because the samples (i.e. the variants) are dependent, since they are tested on the same systems. The significance level (α-level) we will use is the commonly used level 0.05.

Unfortunately, the weighted combined algorithm is non-deterministic. In order to mitigate this and get more results that are more reliable we ran the algorithms repeatedly and per repository calculating new averages until new runs produced only difference in the decimals.

D. Validity threats

We deployed the validity threat model by Runeson and Höst where we classify validity threats into the categories construct,

TABLE II
SUBSYSTEM RECOVERY ACCURACY OF WCA-VARIANTS

System	Combined	Separated	Names-used Only	User-names Only
AutoMapper	47.0%	49.3%	46.9%	43.9%
dotnet	30.3%	29.0%	29.0%	34.5%
DotNetOpenAuth	22.1%	22.6%	22.1%	22.9%
EntityFramework	44.5%	45.0%	43.0%	38.5%
fluentmigrator	43.8%	41.6%	39.9%	36.9%
FluentValidation	40.1%	39.8%	37.0%	38.8%
MonoGame	40.8%	44.2%	48.5%	49.7%
Nancy	42.1%	41.4%	41.7%	38.5%
octokit.net	46.0%	41.7%	52.5%	48.2%
OpenRA	43.5%	43.9%	47.6%	43.1%
ShareX	26.6%	24.7%	26.9%	25.7%
SignalR	23.2%	21.6%	20.4%	22.7%
SparkleShare	46.6%	42.7%	47.1%	43.9%
Wox	23.0%	23.3%	16.1%	18.9%
Average	**37.1%**	**36.5%**	**37.1%**	**36.2%**

internal, and external validity [21].

1) Construct validity: To our knowledge, neither WCA nor MojoFM had been implemented in C# before, and thus we had to implement them ourselves.

Unfortunately, since we were not able to benchmark our WCA implementations against the same dataset as Maqbool and Babri [4], we did run tests on their examples to ensure correctness of our implementation. Furthermore, we used a declarative code style tightly following set theory notation to make it easier to validate code against formulas found in papers.

MojoFM was implemented in Java by its creators, Wen and Tzerpos [5], and to ensure that our implementation of it was done correctly we translated their implementation to C#.

2) Internal Validity: One factor that could affect the results of our tests is the cutting algorithm used to flatten the hierarchical tree so that it can be used with MojoFM for comparison. Depending on where the tree is cut the resulting clusters vary both in size and amount. Researchers use different approaches when it comes to finding where to cut, and research has been done to find the best cutting point [22]. Garcia et al. chose to use the ground truth to find the optimal cut, which, while valid for their comparison, we find counterproductive, since usually the ground truth is unavailable when reverse-engineering a system [15]. Maqbool and Babri chose to use three different cuts at 65%, 75%, and 85% of all entities [18] and compared the different cuts.

We chose to cut at the middle of the tree. While this certainly results in less accuracy than using a more optimal cut, we would like to argue that it does not matter for our results since we are not looking for high recovery accuracy. Instead, we are comparing different variants of WCA, and as long as we cut the trees in the same way we can compare the results to find the most accurate variant.

3) External Validity: While we implemented our tests in C#, the algorithm is language independent and should provide equal results should it be implemented in another language. In regards to the systems used for testing, we tried to make a selection with as many different types and sizes of systems to maximize the diversity of the selection. Additionally, we are not related to the systems being tested on in any way, which should reduce bias in the results.

VI. RESULTS

The results of the tests can be found in table II. The first columns is the system tested on, the second column contains the results when user-names and names-used are combined, while the third column contains the results when they are separated. The fourth column contains the results when matching only by names-used, and finally the fifth column is the results when matching only by user-names.

A. The effect on accuracy when including user-names

Comparing the second and third columns of table II to the fourth and fifth, we can see that in some cases including both names-used and user-names results in a better accuracy, while in others including only one of the two nets the better result. The largest difference can be found in octokit.net, where the difference between the separated variant and the variant using only names-used is as much as 10.8%. Despite that, the average results of the variations show that the variants are mostly equal, with the largest difference in average accuracy being 0.9%.

Results of paired t-tests:

- Combined and Names-used: t(13) = 0.046534, p = 0.9636
- Combined and User-names: t(13) = 0.85666, p = 0.4071
- Separated and Names-used: t(13) = -0.49975, p = 0.6256
- Separated and User-names: t(13) = 0.28957, p = 0.7767

The paired t-tests proves that while the variant used can have a impact on the results, it is not a significant factor in the results. Therefore, we cannot reject our first null hypothesis.

> *OBSERVATION #1:* Including user-names in addition to names-used does *not* increase the accuracy of hierarchical clustering algorithms.

TABLE III
EFFECTS OF DATA LOSS

System	100% Dependencies	50% Dependencies	25% Dependencies
AutoMapper	46.9%	35.2%	34.8%
dotnet	29.0%	5.9%	5.0%
DotNetOpenAuth	22.1%	11.5%	11.2%
EntityFramework	43.0%	32.4%	32.7%
fluentmigrator	39.9%	50.0%	52.2%
FluentValidation	37.0%	34.3%	25.0%
MonoGame	48.5%	41.0%	35.9%
Nancy	41.7%	49.5%	50.6%
octokit.net	52.5%	25.7%	26.1%
OpenRA	47.6%	19.1%	21.2%
ShareX	26.9%	35.7%	24.9%
SignalR	20.4%	13.1%	11.3%
SparkleShare	47.1%	55.4%	48.5%
Wox	16.1%	16.7%	18.2%
Average	**37.1%**	**30.4%**	**28.4%**

B. The effect of separating names-used and user-names

Looking at the second and third columns of table II we can see that the difference between the algorithm when names-used and user-names are combined and when they are separated is minor. For some systems the generated architecture has a higher accuracy when names-used and user-names are combined, and for others it is higher when they are separated. The greatest difference can once again be found in octokit.net, where the combined algorithm gave a 46.0% accuracy versus 41.7% from the separated, a difference of 4.3%, but on average the algorithm has only a 0.6% higher accuracy when combined, a negligible difference. The results of a paired t-test proves this: $t(13) = 1.0926$, $p = 0.2944$. Thus, we cannot reject our second null hypothesis either.

> *OBSERVATION #2:* Separating names-used and user-names when calculating cluster similarity does *not* increase the recovery accuracy of hierarchical clustering algorithms.

VII. DISCUSSION

From our test results we can see that, contrary to our hypotheses, separating names-used and user-names in the weighted combined algorithm does not have a significant impact on the accuracy of the algorithm. We can also see that the variants including both feature types does not give improved results over the names-used only variant. This could possibly be the reason why modern papers like [4] and [15] are not explicit about what kinds of features they include.

Our first assumption was that shared user-names is not a criteria people seem to use when grouping software components. However, when we tested the opposite (i.e. relying only on user-names and ignoring names-used) and also got similar results, we realized this assumption was incorrect. We believe it is rather a case of having a surplus of information in the relation graph. Although one may want to argue that taking more information into account during clustering should increase recovery accuracy, that does not seem to be the case, at least when it comes to adding names-used in addition to user-names features.

A. Effects of data loss

Because the algorithm exhibited this nature of ”stability” where it returns the same result when given less information, we decided to test how the algorithm is affected when removing a percentage of the features. We ran tests on the names-used only variant where we removed 50% and 75% of the features. The removed features were chosen randomly for each iteration, and we ran it for 100 iterations and averaged the results. The results are presented in table III. As can be seen from the results, the algorithm does not provide the same amount of stability when removing features within the variants as when ignoring one feature kind. While in most cases the accuracy drops significantly as expected, it increases in a few, with an increase of over 10% in one case. This is probably related to the phenomenon that too many features can be detrimental to accuracy [23].

B. Similarity of decompositions

It is important to note that whenever we say the ”same result” we mean the same recovery accuracy. This does not necessarily mean the actual recreated clusters are the same, but that the clusters recovered are equal in accuracy towards the ground truth data. For example in the case of DotNetOpenAuth where both variants get around 22% recovery accuracy, it does not necessarily mean that they those 22% represent the same samples of the ground truth data. We therefore ran tests measuring the MojoFM similarity between the decompositions retrieved from the names-used only variant and the user-names only variant.

As can be seen in the results in table IV, the two variants (user-names only and names-used only) produce a similarity above 50% for all systems, which means that for each system the architectures retrieved by the two variants are similar. In other words, we can conclude that most classes that share

TABLE IV
SIMILARITY BETWEEN THE RESULTING ARCHITECTURES OF NAMES-USED ONLY AND USER-NAMES ONLY VARIANTS

System	Similarity
AutoMapper	80.2%
dotnet	82.4%
DotNetOpenAuth	74.6%
EntityFramework	78.8%
fluentmigrator	69.1%
FluentValidation	75.9%
MonoGame	84.7%
Nancy	71.2%
octokit.net	87.6%
OpenRA	56.3%
ShareX	72.5%
SignalR	92.2%
SparkleShare	53.7%
Wox	54.5%
Average	**73.8%**

names-used also share user-names; hence ignoring one of them gives the same result.

C. Transitivity of agglomerative algorithms

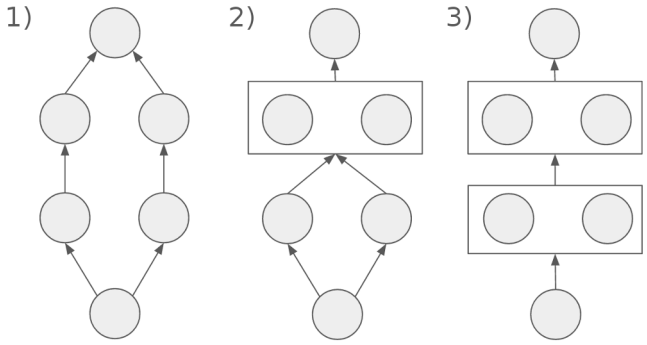

Fig. 3. The steps involved when merging clusters

One possible explanation for the similarity of the architectures generated by the names-used only and the user-names only variants is the transitive nature of how agglomerative algorithms resolve clusters. For example, in fig. 3, step 1 depicts a graph where the upper pair of nodes would be seen similar when only looking at their shared names-used features, whereas the lower pair would only be deemed similar when comparing their user-names features. However, after the algorithm has merged the upper pair into a cluster (step 2), both entities in the lower pair will reference the newly created cluster and share the same names-used feature. Hence the lower pair will actually be turned into a cluster in the last iteration of the algorithm (step 3) even when only names-used feature types are taken into consideration.

Because of this transitive nature of the algorithms, many graphs that may seem like they contain clusters that cannot be found can often be resolved if the sub-graph is part of a bigger more complex graph that is resolved first.

D. Computational cost

Maqbool and Babri and Duda et al. states that more features increases the computational cost of the algorithm, something we have also seen during our tests [18], [23]. Since the combined variant puts weaker requirements for a possible feature to be present (i.e. it can be either user-name or a name-used), the combined variant should make a bigger ratio of the possible features present than the rest of the variants, which should have some impact on performance. However, the separated algorithm should produce a much larger set of possible features, which should by far have the biggest performance impact. The best performing variants should thus be the used-names only and user-names only variants. This means the one direction-only variants can be used with roughly the same accuracy for less computational cost.

VIII. RESEARCH LIMITATIONS AND FUTURE RESEARCH

Our tests has only been done on systems written in C#, and as such we can only conclude that our results hold true for C# systems. While the algorithm itself does not functionally change between different programming languages, different languages have practices and standards that affect the architecture of systems. It is also common that development tools and frameworks impose certain architectural styles or patterns [24].

We personally think that the results would be similar, if not the same, for systems written in other programming languages. One argument that supports this is that the type of subsystem we target to recover has been projects. We would argue that projects is the type of system decomposition that most closely resembles layers in layered architecture [7]. It hence exhibits the behavior for which the phenomenon, as explained in section III, is most likely to occur. Because we have tested in the scenario optimal for our phenomenon to occur but still do not see the phenomenon occurring, it is highly unlikely that it would occur in other scenarios.

Another subject where future research is needed is how names-used and user-names have an effect on hierarchical divisive clustering algorithms. We did not do any tests on divisive algorithm since it was out of scope for our research, but the results might be similar since both agglomerative and divisive algorithms are hierarchical.

Finally, further research should be done on what it is that causes some systems to have higher accuracy when using both names-used and user-names, and others when using only one of them. As seen in our results (table II), for some of the systems the difference in accuracy between the variants is not minor. Finding the cause of these differences could lead to improved clustering algorithms in the future.

IX. CONCLUSION

In the discussion we concluded that classifying features differently depending on relationship direction does not have a significant effect on cluster recovery accuracy. We also concluded that this is not an effect caused by clusters more commonly being grouped based on names-used instead of

user-names, but rather an effect of having a surplus of information in the relation graphs, and that there is a similarity between the decompositions retrieved when including only names-used or only user-names features.

We have provided empirical data that, contrary to our own hypotheses and Schwanke and Platoff's original proposal, shows that one of the feature variants of names-used and user-names can be safely ignored. This gives the same results in recovery accuracy with less computational cost.

ACKNOWLEDGMENT

The authors would like to thank Imed Hammouda for the advice he has given them throughout their research.

REFERENCES

[1] R. W. Schwanke and M. A. Platoff, "Cross references are features," in *ACM SIGSOFT Software Engineering Notes*, vol. 14, pp. 86–95, ACM, 1989. I, II, III, III-A, III-B

[2] E. Eilam, *Reversing: secrets of reverse engineering*. John Wiley & Sons, 2011. II

[3] L. Rokach and O. Maimon, "Clustering methods," in *Data mining and knowledge discovery handbook*, pp. 321–352, Springer, 2005. II

[4] O. Maqbool and H. A. Babri, "The weighted combined algorithm: A linkage algorithm for software clustering," in *Software Maintenance and Reengineering, 2004. CSMR 2004. Proceedings. Eighth European Conference on*, pp. 15–24, IEEE, 2004. II, III-B, IV-A, IV-C, IV-D, V, V-D1, VII

[5] Z. Wen and V. Tzerpos, "An effectiveness measure for software clustering algorithms," in *Program Comprehension, 2004. Proceedings. 12th IEEE International Workshop on*, pp. 194–203, IEEE, 2004. II, IV-D, V-C, V-D1

[6] E. Evans, *Domain-driven design: tackling complexity in the heart of software*. Addison-Wesley Professional, 2004. III-B

[7] F. Buschmann, K. Henney, and D. Schimdt, *Pattern-oriented Software Architecture: On Patterns and Pattern Language*, vol. 5. John wiley & sons, 2007. III-B, VIII

[8] J. Lakos, *Large-scale C++ software design*. Addison-Wesley Reading, 1996. III-B

[9] K. H. Bennett and V. T. Rajlich, "Software maintenance and evolution: a roadmap," in *Proceedings of the Conference on the Future of Software Engineering*, pp. 73–87, ACM, 2000. IV

[10] M. H. B. Osman *et al.*, *Interactive scalable condensation of reverse engineered UML class diagrams for software comprehension*. PhD thesis, Leiden Institute of Advanced Computer Science (LIACS), Faculty of Science, Leiden University, 2015. IV

[11] M. Shtern and V. Tzerpos, "Clustering methodologies for software engineering," *Advances in Software Engineering*, vol. 2012, p. 1, 2012. IV-A

[12] S. Mancoridis, B. S. Mitchell, Y. Chen, and E. R. Gansner, "Bunch: A clustering tool for the recovery and maintenance of software system structures," in *Software Maintenance, 1999.(ICSM'99) Proceedings. IEEE International Conference on*, pp. 50–59, IEEE, 1999. IV-A

[13] J. Garcia, D. Popescu, C. Mattmann, N. Medvidovic, and Y. Cai, "Enhancing architectural recovery using concerns," in *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering*, pp. 552–555, IEEE Computer Society, 2011. IV-A, IV-D

[14] V. Tzerpos and R. C. Holt, "Acdc: An algorithm for comprehension-driven clustering," in *wcre*, p. 258, IEEE, 2000. IV-A

[15] J. Garcia, I. Ivkovic, and N. Medvidovic, "A comparative analysis of software architecture recovery techniques," in *Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on*, pp. 486–496, IEEE, 2013. IV-A, IV-D, V, V-D2, VII

[16] T. A. Wiggerts, "Using clustering algorithms in legacy systems remodularization," in *Reverse Engineering, 1997. Proceedings of the Fourth Working Conference on*, pp. 33–43, IEEE, 1997. IV-B, IV-D

[17] P. Andritsos, P. Tsaparas, R. J. Miller, and K. C. Sevcik, "Limbo: Scalable clustering of categorical data," in *EDBT*, pp. 123–146, Springer, 2004. IV-B

[18] O. Maqbool and H. A. Babri, "Hierarchical clustering for software architecture recovery," *Software Engineering, IEEE Transactions on*, vol. 33, no. 11, pp. 759–780, 2007. IV-B, V-D2, VII-D

[19] N. Anquetil and T. C. Lethbridge, "Experiments with clustering as a software remodularization method," in *Reverse Engineering, 1999. Proceedings. Sixth Working Conference on*, pp. 235–255, IEEE, 1999. IV-C

[20] B. S. Mitchell and S. Mancoridis, "Craft: a framework for evaluating software clustering results in the absence of benchmark decompositions [clustering results analysis framework and tools]," in *Reverse Engineering, 2001. Proceedings. Eighth Working Conference on*, pp. 93–102, IEEE, 2001. V-C

[21] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical software engineering*, vol. 14, no. 2, pp. 131–164, 2009. V-D

[22] C. Y. Chong, S. P. Lee, and T. C. Ling, "Efficient software clustering technique using an adaptive and preventive dendrogram cutting approach," *Information and Software Technology*, vol. 55, no. 11, pp. 1994–2012, 2013. V-D2

[23] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern classification*. John Wiley & Sons, 2nd ed., 2000. VII-A, VII-D

[24] A. Albani, S. Overhage, and D. Birkmeier, "Towards a systematic method for identifying business components," in *Component-Based Software Engineering*, pp. 262–277, Springer, 2008. VIII