IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2024.0429000

Refining Software Clustering: The Impact of Code Co-Changes on Architectural Reconstruction

STANA ADELINA DIANA[1] and SORA IOANA[2]
[1]Computer Science and Engineering Department "Politehnica" University of Timisoara, Romania (e-mail: stana.adelina.diana@gmail.com)
[2]Computer Science and Engineering Department "Politehnica" University of Timisoara, Romania (e-mail: ioana.sora@cs.upt.ro)

Corresponding author: Stana Adelina Diana (e-mail: stana.adelina.diana@gmail.com).

ABSTRACT Version control systems primarily offer support for tracking and managing changes in software code, but they can also provide a great deal of additional information about the managed software. Changes in multiple software entities simultaneously can imply that these entities are connected to one another. Software entities that change at the same time (code co-changes) are treated as a distinct category of software dependencies. In some cases, these co-changes are used together with other types of dependencies, such as structural dependencies or lexical dependencies, to enhance the understanding of a software system.
This paper proposes using code co-changes in software clustering for architectural reconstruction. Structural dependencies are the most commonly used type of dependencies in software clustering for architectural reconstruction, so we will use clustering solutions obtained from structural dependencies as the baseline for our evaluations. Our approach will be applied to five open-source projects found on GitHub. For each of the projects, we will compare and evaluate the clustering obtained by using only co-changes and the clustering obtained by using co-changes combined with structural dependencies with the baseline clustering generated solely from structural dependencies.

INDEX TERMS Architectural reconstruction, code co-changes, logical dependencies, software clustering, software dependencies, versioning system.

I. INTRODUCTION

Software systems often face a lack of documentation. Even if there was original documentation at the beginning of development, over the years it may become outdated or lost. Additionally, the original developers may leave the company, taking with them knowledge about how the software was designed. This situation challenges the teams when it comes to maintenance or modernization. In this context, recovering the system's architecture is essential. Understanding the system's architecture helps developers better evaluate and understand the nature and impact of changes they need to make. One technique to aid in reconstructing the system architecture is software clustering. Software clustering involves creating cohesive groups (modules) of software entities based on their dependencies and interactions.

Among the dependencies that can be used for software clustering are structural dependencies (relationships between entities based on code analysis), lexical dependencies (relationships based on naming conventions), and code co-changes/logical dependencies (relationships between entities extracted from the version control system), among others. Combining multiple types of dependencies, rather than relying on just one type, can be a good approach to generate better results. However, it requires fine-tuning the amount of dependencies used from each category and scaling the coefficients attached to them. Combining dependencies without considering these aspects might lead to results that are less effective than using an individual type of dependency alone.

In this paper, we assess whether using structural dependencies combined with logical dependencies can provide better results than using each type of dependency alone. The structural dependencies are used as they are extracted from static code analysis. The logical dependencies are filtered co-changes from the version control system. The reason behind filtering the co-changes and not using them as they are in the versioning system is to enhance their quality and make them easier to combine with structural dependencies, as their size can outnumber the structural dependencies [1].

To evaluate the results, we generate software clustering on five open-source projects and use two types of metrics for comparison. One of the metrics is MQ (Modularization Quality), which evaluates the modularization quality based on

the interaction between the modules and does not require any additional input besides the clustering result [2]. The other is the MoJo (Move and Join) metric, a commonly used metric for evaluating the similarity between two different software clustering results [3]. For this metric, we manually generate a base of comparison for the clustering result and compare it against this baseline.

In Section II, we review the related work and previous studies that used various dependencies for software clustering and their metrics for evaluation. Section III details the workflow and implementation of our approach, including the extraction and filtering of dependencies, and the clustering algorithm used. The results of our experiments on five open-source projects are presented in Section IV. Section V evaluates our results by using the Modularization Quality (MQ) metric and the Move and Join (MoJo) metric. We also manually analyze some of the clustering solutions. Finally, Section VI contains our conclusions, findings, and potential directions for future work.

A. ABBREVIATIONS AND ACRONYMS

The following abbreviations and acronyms are used throughout this article:

- **LD**: Logical Dependencies
- **SD**: Structural Dependencies
- **MQ**: Modularization Quality Metric
- **MoJo**: Move and Join Metric

Logical Dependencies (LD) refer to the relationships between software entities that have been extracted and filtered from the versioning system. If these entities are not filtered, they are simply referred to as co-changes. Structural Dependencies (SD), on the other hand, refer to the relationships between software entities extracted from static code analysis. Modularization Quality Metric (MQ) and Move and Join Metric (MoJo) are metrics used to evaluate software clustering results.

II. RELATED WORK

III. WORKFLOW AND IMPLEMENTATION

To achieve our goal of evaluating how the quality of clustering solutions is impacted by logical dependencies, we developed a Python tool capable of using any type of dependency, either alone or combined with other types of dependencies, as long as they are provided in CSV format. The tool clusters and evaluates software clustering solutions using either the MQ metric or the MoJo metric. In the following subsections, we present how we obtain the structural dependencies and logical dependencies used, the type of clustering algorithm we use, and the tool's workflow.

A. STRUCTURAL DEPENDENCIES

The structural dependencies are obtained using a tool from our previous research. While the tool primarily exports the key class ranking of a software system, it also has the capability to export the data in CSV format. The exported dependencies are directed and weighted based on the type of dependency they represent.

B. LOGICAL DEPENDENCIES

We refer to logical dependencies as the filtered co-changes between software entities. A co-change occurs when two or more software entities are modified together during the same commit in the version control system. Co-changes indicate that these entities are likely related or dependent on each other, directly or indirectly.

There is a degree of uncertainty associated with co-changes. Compared to structural dependencies, where the presence of a dependency is certain, co-changes are less reliable. For example, if the system was migrated from one version control system to another, the first commit will include all the entities from the system at that point in time. Should we consider all these entities as related to one another in this case? This would introduce false dependencies and reduce the likelihood of achieving accurate results when combining them with more reliable types of dependencies.

Even if we address the issue of the first commit, it can still happen that a developer resolves multiple unrelated issues in the same commit (even though this is not recommended by development processes).

To solve this problem, in our previous works, we have refined some filtering methods to ensure that the co-changes that remain after filtering are more reliable and suitable for use with other dependencies or individually.

C. LOUVAIN CLUSTERING ALGORITHM

D. TOOL WORKFLOW OVERVIEW

IV. RESULTS

V. EVALUATION

VI. CONCLUSION

REFERENCES

[1] Ajienka, Nemitari & Capiluppi, Andrea. (2017). Understanding the Interplay between the Logical and Structural Coupling of Software Classes. Journal of Systems and Software. 134. 10.1016/j.jss.2017.08.042.
[2] S. Mancoridis, B. Mitchell, C. Rorres, Y. Chen, and E. Gansner, "Using automatic clustering to produce high-level system organizations of source code," in Proceedings. 6th International Workshop on Program Comprehension. IWPC'98 (Cat. No.98TB100242), 1998, pp. 45–52.
[3] V. Tzerpos and R. C. Holt, "MoJo: a distance metric for software clusterings," Sixth Working Conference on Reverse Engineering (Cat. No.PR00303), Atlanta, GA, USA, 1999, pp. 187-193, doi: 10.1109/WCRE.1999.806959.
[4] Stana, Adelina-Diana & Şora, Ioana. (2023). Logical dependencies: Extraction from the versioning system and usage in key classes detection. Computer Science and Information Systems. 20. 25-25. 10.2298/CSIS220518025S.

FIRST Add text here

SECOND Add text here