Remittances Review
August 2024,
Volume: 9, No: 4, pp.770-780
ISSN: 2059-6588(Print) | ISSN 2059-6596(Online)

Received : 15 February 2024, Accepted: 10 July 2024
DOI: https://doi.org/10.33282/rr.vx9i2.44

A SYSTEMATIC ANALYSIS ON SOFTWARE ARCHITECTURE RECOVERY TECHNIQUES

Usama Hafeez[1], Muhammad Kaleem[2],Muhammad Azhar Mushtaq[2] , Shahid Khan[3], Sheraz Butt[4], Ammad Ahmed[5] , Dr.Sadaqat Ali Ramay[6] ,Sayyid Kamran Hussain[6]

[1]Department of Software Engineering, University of Sargodha, Sargodha, Pakistan

[2] Department of Information Technology, University of Sargodha, Sargodha, Pakistan

[3]Senior SolutionArchitect, P.O.Box 357, Dammam SaudiArabia

[4]Senior System Analyst, Department of Social Services, South Carolina, USA

[5]Product Design (UX/UI) manager Dubai,UAE

[6]Department of Computer Science, Faculty of Science and Technology, TIMES Institute, Multan, Pakistan.

Abstract

Software architecture describes the components and their interactions inside a software system. Rapid iteration and frequent releases have become crucial in today's software industry. As a result, there has been a breakdown in the software's architecture due to a lack of careful planning and documentation during its development. Software architecture recovery refers to the process of reassembling the architecture of a software system from its implementation-level artifacts. In order to automatically reconstruct software architectures from software implementations, many different approaches have been proposed. These methods and tools include reverse engineering, static analysis, and dynamic analysis. Software architects, designers, and maintenance engineers have access to a powerful tool in the form of these methodologies, which they may utilize to assist a variety of software-development processes and assure the long-term sustainability of software systems. This study's main goal is to examine existing software architecture recovery methods.

Keywords: Software architecture recovery, software reconstruction, software components recovery.

Remittances Review
August 2024,
Volume: 9, No: 4, pp.770-780
ISSN: 2059-6588(Print) | ISSN 2059-6596(Online)

I. INTRODUCTION

It is essential to have an understanding of software in order to keep it running smoothly and to perform any necessary maintenance on it. Insufficient paperwork and a lack of understanding with the software present the greatest dilemma during the recovery phase [1]. This is because the cost of manual recovery is high, and it takes a significant amount of time. As a result of this, there are currently efforts being made to lighten the load[2]. As more time passes, the documentation has either stopped being regularly updated or is completely absent. Numerous research efforts have been focused on reconstructing the software's architecture [3], and it has been the subject of these efforts. The recovered architecture can be utilized for variety of purposes, including the analysis of the overall structure of the software, the identification of architectural improvements and deterioration [4], and so on.

In the event that software evolves, less attention being paid to the architecture of software will result in enormous costs associated with the redevelopment of the product from scratch. It is essential for the architecture of the software in question to be both known and flexible in order to encourage its evolution. It is possible that performance will suffer if modifications that violate the architecture of the system are implemented. Using the syntactic component of the program, a large variety of methodologies have been presented over the course of the years in order to aid in recovering the structure of such software systems [5].

In software architecture recovery, the whole system is broken down into its component parts so that they can be understood individually. In the research that has been done on the subject, terms such as decomposition, re-modularization, splitting, clustering, and reconstruction have been used to talk about the different ways that software architecture can be rebuilt. The following four steps are taken to restore the structure of the software: highlighting a software entity; computing the degree of similarity between entities; clustering; evaluating [6].

Software product line construction produces groups of software products with greater quality and lower development costs and time to market. By performing an analysis of the similarity and variance of the different types of products that are to be developed and by recycling common components as much possible during the process of actually developing the product variants, Software Product Line Engineering (SPLE) is able to overcome the challenges that are presented by the clone-and-own methodology. So, it is logical to consider moving the product variations that were developed based on the clone-and-own strategy can be utilized for further maintenance and development of the product. These product variants are also known as product families [6].

This survey is being carried out with the intention of gaining a better understanding of software architecture loss as well as the solutions that have been proposed to reconfigure the architecture of certain software by making use of recovery techniques. The remaining sections of the paper are as follows: Section II discusses related work; Section III details the challenges; Section IV features a conclusion of the paper; and Section V offers discussion of possible future work.

II. CHALLENGES AND ISSUES OF SOFTWARE ARCHITECTURE RECOVERY

Software architecture recovery refers to the process of re-creating the architecture of a source code using the system's present implementation-level software artifacts and documentation as the primary sources of information. Despite of these advancements, there are still a great many obstacles that need to be overcome to render source code recession a more effective solution. These obstacles include the following:

Remittances Review
August 2024,
Volume: 9, No: 4, pp.770-780
ISSN: 2059-6588(Print) | ISSN 2059-6596(Online)

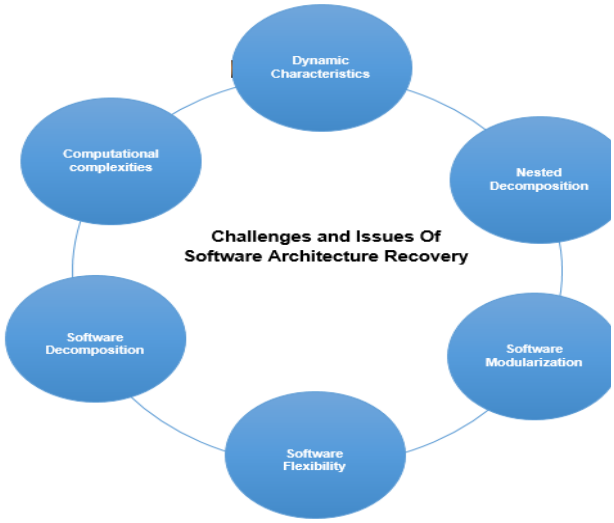

Figure 1. Challenges and issues of software architecture recovery

1. Software systems can become more complicated over time; sometimes this is due to the emergence of computational complexities, which makes it difficult to comprehend the connections between the various components and the ways in which they collaborate.[10].
2. Software clustering methods produce large decompositions as visualization output. Consequently, it is difficult to design a user interface that can effectively convey a software decomposition to a software engineer. Decomposition of software should be linked to its source code, documentation, and so on, and the user interface should make it easy to navigate through the results. Better visualization of software clustering results would increase the effectiveness of software clustering methods. [32].
3. It can be challenging to encapsulate and comprehend the dynamic characteristics of a software package, such as the interactions between its components. Prior to making any additional key changes, the institutional views must be as precise as possible. Deployment views, for instance, continue to depend strongly on precise cognize and component configurations [1].
4. The flexibility of conventional recovery methods can be improved by grouping the elements before extraction,moreovereventually results strategies with higher scalability could exert precision issues for each method [33].
5. Operations known as refactoring and re-modularization can be carried out on software architecture in order to remove complexities that have been caused by software evolution. As a consequence of this, it can be challenging to modularize system classes while preserving the optimal balance that should exist between cohesion and coupling [11].
6. The process of cluster discovery, which includes both the phase of computing similarity and the phase of creating clusters, is one that can be improved by either a new source code clustering method or developing a new technique for computing resemblance coefficients. A nested decomposition can be generated by a software clustering method however, there is not yet an optimization-based software clustering method that can generate a nested decomposition [32].

III. Taxonomy of Architecture Recovery Techniques

Clustering algorithms, behavior-based methods, and filtering methods all play roles in architecture recovery methods. Most filtering methods also are premised on clustering algorithms because they only give us partial information about the software system's architecture to begin with, whereas in cluster formation, the groups formed of familiar elements typically reveal the full picture. Behavior-based techniques are being incorporated into existing clustering algorithms because the attempt necessitated by locating the architecture of a current applications using a clustering method is lower than that provided by the behavior-based techniques.[7].

Characterizing software architecture from code manuscripts is a time-consuming process, and it can be difficult to know how much weight to give to structural and semantic information such as comments and identifier names. Low-quality code, a lack of software dependency info,

Remittances Review
August 2024,
Volume: 9, No: 4, pp.770-780
ISSN: 2059-6588(Print) | ISSN 2059-6596(Online)

and structural chaos all work against successful architectural recovery [8]. This issue may be resolved by locating and removing textual anomalies in software code elements which could cause an incorrect architecture recovery [9].

Conceptual conformity (CC) is a score indicating how closely the subject matter of the code and the package are related. According to [8] a textual anomaly is a chunk of code that CC deems unrelated to the rest of the package's contents. Semantic-information-based architectural recovery [10] allows for the conceptual dissection of software systems. However, the precision of the decomposition is affected by the textual efficiency of the source code. It is possible to achieve a higher level of precision in the architecture recovery process by omitting textual anomalies.

Coevolving software components are said to be evolutionarily coupled because of the underlying communication between them. Software architecture can be retrieved through experimentation with the coupling of evolution [11]. Software Architecture Recovery (SAR) uses the dependency graph as its input mechanism. Many kinds of communication between modules of software are represented by these graphs. It is the developers' intention, during implementation, to keep the overall level of cohesion and coupling between modules reasonable[12].

There are a number of methods proposed for automatically recovering software architecture from code. Utilizing symbol dependencies on recovery strategies by include them as inputs, these are more precise then input dependencies whereas input dependencies can improve prior research [1].

An appropriate ground-truth architecture takes an average of 80–90 hours of work from a competent individual [13] , and that's just for average-sized projects. Having a firm grasp of ground truth architectures helps us better understand designs and develop more effective recovery methods. The Cacophony technique [14]. involves recovering and reverse-engineering a software system by hand using a metamodel. Rigi [15] and ShriMP[16] are indeed the tools to use when analyzing and visualizing software dependencies. To manually arrange components that are similar, you will need a developer who has an in-depth understanding of the project.

Clustering is more challenging with categorical data because there is no natural distance between data values. The highly efficient hierarchical clustering algorithm, which would be premised upon an Information Bottleneck structure for figuring out how much relevant information is kept when clustering is LIMBO, does have the benefit of being able to make clusters of different sizes in a single run. This lets you choose between speed and quality[20].

Several articles analyzing and contrasting various methods for recovering lost architectural information make up the relevant literature. However, attention was paid to only a small subset of clustering algorithms. The purpose of this paper is to provide a comprehensive overview of software architecture recovery approaches.

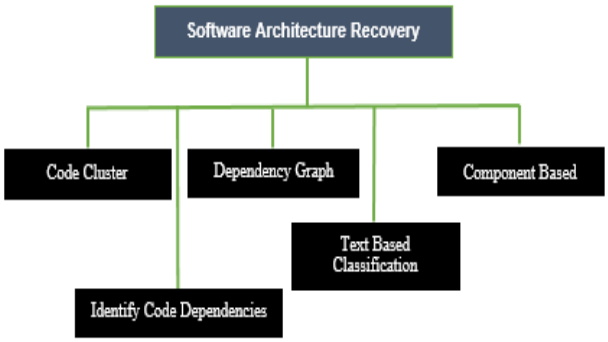

Figure 2. Taxonomy of software architecture recovery

A. Code cluster

Decomposing a complex software system into smaller subsystems is made easier with the aid of code pattern/cluster, which does so by recovering components via patterns. It achieves its goal in that the clusters are appropriately named, their architectures are based on established patterns that developers relied upon when creating software artefacts, and there are no superfluous elements within any given cluster.

Structural and textual information can help deliver software architecture from code. Textual

anomalies in source code artifacts hinder architecture recovery. Conceptual Conformity compares two source code and package latent topic distributions. Removing semantic outliers to prevent architecture recovery errors. Semantic information-based architecture recovery deconstructs software systems. Filtering textual anomalies improves decomposition accuracy [8].

Cluster ensembles guide software architecture module recovery. Extracting software facts from the software repository. Base clustering, which are recovered module views, are then created from the extracted facts. Meta-Clustering Algorithm (MCLA), HyperGraphs Partitioning Algorithm (HGPA), Hybrid Bipartite Graph Formulation (HBGF), Evidence Accumulation Algorithm (EA), and Cluster-based Similarity Partitioning Algorithm (CSPA) consensus methods are compared using cluster ensembles. Using a consensus method with multiple base clustering algorithms could improve performance. MCLA and EA performed well in a five-open-source project consensus experiment.[17].

Previous research focused on a system's architecture rather than its cause. RecovAr is a method for inevitably restore design decisions from a project's easily accessible history artefacts. It is not dependent on the architecture of a system, but it does need a way to get static architectural structure from interoperability manuscripts. Architecture Recovery, Change, And Decay Evaluator (ARCADE) measures architectural change and uses architecture-recovery methods. Algorithm for Comprehension-Driven Clustering (ACDC) and Architecture Recovery using Concerns (ARC) recover components from code [10].

Most of the suggested methods use hierarchical clustering algorithms to reconstruct the original layout of a piece of software. In this refined approach to hierarchical clustering, we employ a LIMBO-based fuzzy hierarchical clustering algorithm. To generate a cluster and enhance its accuracy, this algorithm first helps in extracting the knowledge that describes the system for clustering, then initializes weights to the information extracted [18].

Selecting the best software clustering method to aid in deciphering a complex system is challenging. The efficiency of an algorithm can depend on a number of different things, such as the decompositions it generates and the names it gives to its clusters. To evaluate the efficiency of software clustering algorithms, the move or joinalso known as MOJO distance is used. This distance is the fewest move or join operations needed to transform cluster A into cluster B, or vice versa.[19].

B. Dependency graph

Dependency graphs show how various software components are interconnected. Assuming that developers share this objective during design and implementation, these modules are grouped together to achieve a nearly best solution of cohesion and coupling.

Legacy software may lack documentation. Manual documentation recovery is costly. Domain experts work hundreds of hours. Module dependencies derived from source code have dominated module view construction research. These dependencies are usually graphed. If coupling is measured for recovery architecture, the graph's segments can be balanced. File inclusion dependency graphs provide ground-truth architectures. Dependency graphs are input to the module. Evolutionary coupling has been tested for software architecture recovery [11].

To understand large software systems, program modularization and refactoring are used. These algorithms create software architecture by breaking the software system's raw data into relatively small, more manageable modules. The source code artefact dependency graph is not used in these methods. Graph-based modularization overcomes modularization's limitations. This algorithm tests 10 Mozilla Firefox folders and 4 other applications. The proposed algorithm generates reusability that is nearer to the specified directory than other algorithms [21].

Directory paths help recover architecture from software design information. Directories are incorrect due to development and design inconsistency. A file-level dependency graph is used to group intra and inter coupling documents

within a single repository into submodules and generate a submodule-level dependency graph. The results suggest this method could boost recovery efficiency and effectiveness. This technique improves recovery performance on small projects because prevailing configurations work well with a few software entities [22].

C. Text based classification

Software systems often lack current architecture documentation. Concern-oriented architectural recovery methods have been developed to address this issue. RELAX, a new text classification-based concern-based recovery method, solves these problems. Text-based classification for architectural information extraction. These methods are limited by the classifier's extensive architecture knowledge [23].

From the source code, the RELAX software architecture recovery process can achieve a visual and textual concern-based architectural view of the software system. Some of them fail to meet deadlines because they lack recovery data. Participants' experiences and opinions show that RELAX is helpful in speeding up the beginning of maintenance, and it could form the base for future techniques that specifically support the evolution with a focus on maintenance[24].

Using a Nonparametric classification model and an Orphan adoption classifier, this iterative source code recovery technique is driven by code changes to update the original architectural documentation. In contrast to more standard approaches, our method takes into account code resemblance all through evolution and keep updating the prior methodology based upon that changed spots as opposed to performing a full cluster analysis or classification [25].

D. Component based techniques

By recovering or highlighting packages and classes, component-based techniques are used to reconstruct the architecture, which then contributes to the production of UML package and class diagrams.

In software product families, it is common practice to recover the structure of a software product line by studying the products in the family that were created using the clone-and-own approach, which involves recycling strategies used in previously developed products. Classes that are clones of other classes, classes that are clones of modifications to other classes, and classes that are unique to the product itself are all dissected and analyzed. The generated PLA's architectural elements are mapped into a set of centralized packages, with their degrees of similarity and variability represented by class diagrams. Because of the recovered PLA, we can now switch from our current clone and method to one based on SPL [6].

Pre-planned standardized reusability of large-grained software artefacts increases software productivity and reduces development costs in Production Line of Software Engineering. Reverse engineering product variant architecture. We want to find architectural element variant differences and dependencies. Identifying architecture variability from each product does this. Thus, we find component-based architecture in each product's object-oriented source code. Healing begins here. To identify architecture and design variability in component-based architectures, we recognize component variants with similar functionalities [26].

The primary phase of developing complex systems is now the conceptual model and reflection of software architectures. There are many advantages to representing software architecture at all stages. Removing a component based architectural style from an existing object-oriented system is the primary focus of ROMANTIC. The primary goal of this strategy is to suggest a method for architecture recovery that is only partially automated, and which relies on the textual and functional properties of software product concepts [27].

Software product families use the line of software products approach for systematic reuse. Anomalous variants can become very different from their predecessors, making line of products model from existing framework variants difficult. PLA recovery with outlier variants complicates architectural decisions. Formal concept analysis identified outliers. Threshold analysis decreased

the number of an exclusive components while retaining the recovered PLA variants [28].

E. Identify code dependencies

Several methods exist for instantaneously recovery architecture from code. Understanding requires extensive research and comparison. Utilizing symbol relationships on mitigation strategies by including function, method, and variable names as inputs and is more accurate than input relationships. Input dependencies help previous research. The studies show how dynamic bindings resolution, dependency granularity, and direct or transitive dependencies affect recovery technique accuracy and scalability. [1].

Many methods exist to instantaneously retrieve architecture from software implementation. Installing something requires a third-party code dependency. External dependencies affect the application, but it's hard to make software without them. We retrieve the majority of the system's main features from its header files, which are include dependencies. Dependency graphs show symbol relationships among project tasks and outside activities that must be scheduled. We assessed SAR technique dependencies using MoJoFM and Normalized TurboMQ. [29].

Long-term software development with hundreds of billions of lines of code can incur technical debt from module dependencies. Underutilized dependencies slow down the construction process and increase file size, resulting in poor cohesion. Inconsistent dependencies break software. The project's core modules use third-party libraries, an inconsistent dependency. Programmers may break design guidelines and introduce inconsistencies for short-term gains. CodeSurfer and CppDepend extract structural and behavioral dependencies [30].

MoJoFM is a tool that can be utilized to evaluate the recovered architectures in terms of how closely they resemble the architecture that was initially developed. The formula that describes it is as follows:

$$MoJoFM = (1 - \frac{mno(A,B)}{max\ (mno(\forall A,B))}) \ x\ 100 \quad (1)$$

Where A stands for the recovery architecture, B for the ground truth architecture, and mno(A, B) for the minimum amount of Move and Join operations that must be performed in order to transform A into B.

The shortcomings that were listed above are addressed by the effectiveness metric that is presented in this paper. MoJoFM makes the following features available to its users: (1) The paradox is sidestepped by substituting mno(A, B) for MoJo(A, B) in the ratio of the its formula. This allows the argument to be valid. (2) It makes use of the most advanced algorithm currently available for determining mno (A, B). (3) It determines the actual maximum range to partition B by computing the divisor of its formula.

IV. Discussion and Future Work

To recover software architecture, one must learn and record the structure of a conventional software product. The goal of architecture recovery is to analyze and record the architecture of a software system in order to enhance its maintainability, scalability, and quality. Here, we'll talk about why software architecture recovery is so crucial, what kinds of problems it might cause, and what kinds of solutions have been developed to address those problems. It is possible to recover the design of an existing software system by using certain software architecture recovery methods. These methods are critical for software upkeep and improvement because they provide programmers with the information they need to make corrections and additions. Here we'll take a look at some of the most popular software architecture recovery methods. By analyzing the software system's code using code dependency methodologies, its architectural components may be determined. Dynamic and static analysis are two of these methods. Comparatively, dynamic analysis entails running the code and seeing its results, whereas static analysis just examines the code itself. Techniques for analyzing code dependencies may be used to map out how

different parts of a software system are interconnected with one another.

The structure of a program may be represented graphically with the help of component-based approaches. UML diagrams are one kind of diagram that may be used, but there are other methods as well, such as more interactive representations i.e. heat maps and node-link diagrams. Software architects may benefit from visualization approaches since they allow them to better comprehend the software's architecture and behavior. Architectural patterns in software may be found using pattern detection methods. The model-view-Controller pattern is one example of a very popular design pattern that can be found here. Developers may benefit from pattern detection methods since they illuminate the architecture of a software application and point out potential trouble spots. One way to examine how information moves through an application is through the use of dependency graph approaches. Data corruption and loss are only two examples of problems that these methods may help engineers spot. Developers may benefit from data flow analysis approaches by better comprehending the architecture of the program and how it interacts with other systems. A better software system that is simpler to maintain and enhance over time is the result of these methods being used by developers.

The continuation of this study may go in any one of the few directions in the years to come. The following are some of them:

1. Handling Non-Functional Requirements: Numerous existing approaches concentrate on restoring the functional architecture of software systems; moreover, non-functional specifications such as performance, security, and scalability are gaining significance. Therefore, comprehensive strategies that really can restore non-functional architecture are necessary.
2. Non-Code Artefacts Recovery: However, there is an increasing need to take into account numerous different non-code artefacts as pieces of knowledge for architecture recovery. Some examples of these non-code artefacts include deployment architectures, database schemas, and system logs. Often these architecture recovery techniques presently concentrate their attention on code artefacts.
3. Investigating Various Architectural Patterns and Designs: It is necessary to conduct additional research into the different architectural patterns and styles currently in use, in addition to gaining a deeper comprehension of the methods by which these architectural motifs can be recognized and restored from implementation artefacts.
4. Architecture Recovery Tool: In order to turn the entire process of clustering into a usable tool that researchers, computer programmers, and practitioners can use to conduct additional experiments and gather feedback for use in future enhancements. Integration with production tools and processes can improve the efficiency of architecture recovery. This includes things such as bug trackers, code repositories, and continuous deployment pipelines.
5. Improved Accuracy: Researchers are looking into ways to improve the precision of architecture recovery techniques. One possibility is to implement machine learning algorithms, while another is to consider additional sources of information such as supporting documents, developer remarks, and test cases. Both approaches are currently under investigation.

These problems underscore how important it is to have adequate documentation, use modular design that is well-structured, and have a systematic approach to modelling software architecture.

V. CONCLUSION

This research provides a survey of the various recovery techniques for software architecture. The results show that component-based software architecture recovery techniques can be automated. The techniques that were proposed were predominantly based on the mining of the modules and the connector that were engaged in the architects of the software system. This was

accomplished through the proof of identity of the standard and changeable packages and classes. In the beginning of the software's development process, neither the architectural style nor the design pattern were given much thought.

The goal of the Software Architecture Recovery research area is to inevitably decipher the structure of a software application from its improvement artefacts such as codebase, binary code, and implementation configurations. Over the course of the last few decades, this area of study has developed and matured, yielding a variety of methods and techniques that have been implemented in working software systems.

The process of developing software could be significantly altered as a result of Software Architecture Recovery's potential to make a big difference in the field. For this area of study to continue to advance, it will be necessary to conduct research on both new methods and tools, as well as a thorough understanding of the obstacles and limitations posed by previously established approaches. It is possible, with these efforts, to make Source Code Recovery a practice that becomes more efficient and widely used, and it is also possible to assist architects and developers in better understanding and managing the structure of complex systems of software.

REFERENCES

[1] T. Lutellier et al., "Measuring the Impact of Code Dependencies on Software Architecture Recovery Techniques," IEEE Trans. Softw. Eng., vol. 44, no. 2, pp. 159–181, 2018, doi: 10.1109/TSE.2017.2671865.

[2] S. Scalabrino, G. Bavota, C. Vendome, M. Linares-Vasquez, D. Poshyvanyk, and R. Oliveto, "Automatically Assessing Code Understandability," IEEE Trans. Softw. Eng., vol. 47, no. 3, pp. 595–613, 2021, doi: 10.1109/TSE.2019.2901468.

[3] J. Garcia, I. Ivkovic, and N. Medvidovic, "A comparative analysis of software architecture recovery techniques," 2013 28th IEEE/ACM Int. Conf. Autom. Softw. Eng. ASE 2013 - Proc., pp. 486–496, 2013, doi: 10.1109/ASE.2013.6693106.

[4] P. Behnamghader, D. M. Le, J. Garcia, D. Link, A. Shahbazian, and N. Medvidovic, "A large-scale study of architectural evolution in open-source software systems," Empir. Softw. Eng., vol. 22, no. 3, pp. 1146–1193, 2017, doi: 10.1007/s10664-016-9466-0.

[5] J. Misra, K. M. Annervaz, V. Kaulgud, S. Sengupta, and G. Titus, "Software clustering: Unifying syntactic and semantic features," Proc. - Work. Conf. Reverse Eng. WCRE, pp. 113–122, 2012, doi: 10.1109/WCRE.2012.21.

[6] J. Lee, T. Kim, and S. Kang, "Recovering Software Product Line Architecture of Product Variants Developed with the Clone-and-Own Approach," Proc. - 2020 IEEE 44th Annu. Comput. Software, Appl. Conf. COMPSAC 2020, pp. 985–990, 2020, doi: 10.1109/COMPSAC48688.2020.0-143.

[7] A. Nicolaescu and H. Lichter, "Behavior-based architecture reconstruction and conformance checking," Proc. - 2016 13th Work. IEEE/IFIP Conf. Softw. Archit. WICSA 2016, pp. 152–157, 2016, doi: 10.1109/WICSA.2016.25.

[8] 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. 212467–212477, 2020, doi: 10.1109/ACCESS.2020.3040024.

[9] K. Yang, J. Wang, Z. Fang, P. Wu, and Z. Song, "Enhancing software modularization via semantic outliers filtration and label propagation," Inf. Softw. Technol., vol. 145, p. 106818, May 2022, doi: 10.1016/J.INFSOF.2021.106818.

[10] A. Shahbazian, Y. Kyu Lee, D. Le, Y. Brun, and N. Medvidovic, "Recovering Architectural Design Decisions," Proc. -

2018 IEEE 15th Int. Conf. Softw. Archit. ICSA 2018, pp. 95–104, 2018, doi: 10.1109/ICSA.2018.00019.
[11] A. Saydemir, M. E. Simitcioglu, and H. Sozer, “On the Use of Evolutionary Coupling for Software Architecture Recovery,” 2021 Turkish Natl. Softw. Eng. Symp. UYMS 2021 - Proc., pp. 0–5, 2021, doi: 10.1109/UYMS54260.2021.9659761.
[12] I. Candela, G. Bavota, B. Russo, and R. Oliveto, “Using cohesion and coupling for software remodularization: Is it enough?,” ACM Trans. Softw. Eng. Methodol., vol. 25, no. 3, pp. 1–28, 2016, doi: 10.1145/2928268.
[13] J. Garcia, I. Krka, C. Mattmann, and N. Medvidovic, “Obtaining Ground-Truth Software Architectures,” pp. 901–910, 2013.
[14] J. Favre, A. Team, and L. Lsr-imag, “Ca c Oph o Ny : Metamodel-Driven Software Architecture Reconstruction,” 2004.
[15] M. A. D. Storey, K. Wong, and H. A. Muller, “Rigi: A visualization environment for reverse engineering,” Proc. - Int. Conf. Softw. Eng., pp. 606–607, 1997, doi: 10.1109/ICSE.1997.610428.
[16] J. Michaud, M. A. Storey, and H. Müller, “Integrating information sources for visualizing Java programs,” IEEE Int. Conf. Softw. Maintenance, ICSM, pp. 250–259, 2001, doi: 10.1109/ICSM.2001.972738.
[17] C. Cho, K. S. Lee, M. Lee, and C. G. Lee, “Software Architecture Module-View Recovery Using Cluster Ensembles,” IEEE Access, vol. 7, pp. 72872–72884, 2019, doi: 10.1109/ACCESS.2019.2920427.
[18] Y. Wang, P. Liu, H. Guo, H. Li, and X. Chen, “Improved hierarchical clustering algorithm for software architecture recovery,” Proc. - 2010 Int. Conf. Intell. Comput. Cogn. Informatics, ICICCI 2010, pp. 247–250, 2010, doi: 10.1109/ICICCI.2010.45.
[19] Z. Wen and V. Tzerpos, “An effectiveness measure for software clustering algorithms,” Progr. Comprehension, Work. Proc., vol. 12, pp. 194–203, 2004, doi: 10.1109/wpc.2004.1311061.
[20] P. Andritsos, P. Tsaparas, R. J. Miller, and K. C. Sevcik, “LIMBO: Scalable clustering of categorical data,” Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 2992, pp. 123–146, 2004, doi: 10.1007/978-3-540-24741-8_9.
[21] B. Pourasghar, H. Izadkhah, A. Isazadeh, and S. Lotfi, “A graph-based clustering algorithm for software systems modularization,” Inf. Softw. Technol., vol. 133, p. 106469, 2021, doi: 10.1016/j.infsof.2020.106469.
[22] X. Kong, B. Li, L. Wang, and W. Wu, “Directory-Based Dependency Processing for Software Architecture Recovery,” IEEE Access, vol. 6, pp. 52321–52335, 2018, doi: 10.1109/ACCESS.2018.2870118.
[23] D. Link, P. Behnamghader, R. Moazeni, and B. Boehm, “Recover and RELAX: Concern-oriented software architecture recovery for systems development and maintenance,” Proc. - 2019 IEEE/ACM Int. Conf. Softw. Syst. Process. ICSSP 2019, pp. 64–73, 2019, doi: 10.1109/ICSSP.2019.00018.
[24] D. Link, K. Srisopha, and B. Boehm, “Study of the utility of text classification based software architecture recovery method relax for maintenance,” Int. Symp. Empir. Softw. Eng. Meas., 2021, doi: 10.1145/3475716.3484194.
[25] O. Maqbool and H. A. Babri, “Bayesian learning for software architecture recovery,” 2007 Int. Conf. Electr. Eng. ICEE, 2007, doi: 10.1109/ICEE.2007.4287309.
[26] A. Shatnawi, A. D. Seriai, and H. Sahraoui, “Recovering software product line architecture of a family of object-oriented

product variants," J. Syst. Softw., vol. 131, pp. 325–346, 2017, doi: 10.1016/j.jss.2016.07.039.
[27] S. Chardigny, A. Seriai, M. Oussalah, and D. Tamzalit, "Extraction of component-based architecture from object-oriented systems," 7th IEEE/IFIP Work. Conf. Softw. Archit. WICSA 2008, pp. 285–288, 2008, doi: 10.1109/WICSA.2008.44.
[28] C. Lima, W. K. Assunção, J. Martinez, W. Mendonça, I. C. Machado, and C. F. Chavez, "Product line architecture recovery with outlier filtering in software families: the Apo-Games case study," J. Brazilian Comput. Soc., vol. 25, no. 1, 2019, doi: 10.1186/s13173-019-0088-4.
[29] R. Deshmukh., S. Murarka, R. Agarwal, D. Datta, and P. Borhade, "Software Architecture Recovery Techniques," Int. J. Eng. Adv. Technol., vol. 9, no. 4, pp. 856–859, 2020, doi: 10.35940/ijeat.d8018.049420.
[30] P. Wang, J. Yang, L. Tan, R. Kroeger, and J. D. Morgenthaler, "Generating precise dependencies for large software," 2013 4th Int. Work. Manag. Tech. Debt, MTD 2013 - Proc., pp. 47–50, 2013, doi: 10.1109/MTD.2013.6608678.
[31] M. Shtern and V. Tzerpos, "Clustering Methodologies for Software Engineering," Adv. Softw. Eng., vol. 2012, pp. 1–18, 2012, doi: 10.1155/2012/792024.
[32] T. Lutellier et al., "Comparing Software Architecture Recovery Techniques Using Accurate Dependencies," Proc. - Int. Conf. Softw. Eng., vol. 2, pp. 69–78, 2015, doi: 10.1109/ICSE.2015.136.