International Journal of Engineering and Advanced Technology (IJEAT)
ISSN: 2249 – 8958 (Online), Volume-9 Issue-4, April, 2020

Software Architecture Recovery Techniques

Rajvardhan Deshmukh. Prathamesh Borhade, Samridhi Murarka, Rishav Agarwal, Debajit Datta

Abstract*: *There are many software architecture recovery techniques which has been discovered which automatically recover software architecture from the software implementation. In this project we will propose a research approach for comparing different software architecture recovery techniques. A dependency (code dependency) is a file that something you are trying to install requires. It can be a library of a third-party organization. These dependencies effect the application but it is very hard to make any software without using these external dependencies. But these code dependencies have some disadvantages too. Firstly, we will specify about the code dependencies and their impact on software design. Then we will describe some software architecture recovery techniques. We will take a project (Bash) as our research base and we will apply these recovery techniques to the project. We will use some software testing tools to compare these algorithms (software recovery techniques) with each project.

Keywords*: *Software engineering, clustering algorithm, dependency, architecture, ground-truth.

I. INTRODUCTION

Software architecture: It is the set of structures required to know about the system which also includes elements of software, their properties and relationships among them

Importance of software architecture:

- **Communication among stakeholders:** Software architecture representations helps stakeholders to involve in communication who have an idea about developing systems which are computer based.
- **Early design decisions:** Architecture gives much importance to early decisions on design which has a much greater impact on the upcoming work on software engineering.
- **Graspable model:** IT (software architecture) include a possible graspable model of entities working together along with system that is structured

Revised Manuscript Received on April 25, 2020.

* Correspondence Author

Prathamesh Borhade*, Computer science and engineering, Vellore Institute of Technology, Vellore, India. Email: prathamesh.borhade29@gmail.com

Rajvardhan Deshmukh, Computer science and engineering, Vellore Institute of Technology, Vellore, India. Email: rajvardhan1999@gmail.com

Rishav Agarwal, Computer science and engineering, Vellore Institute of Technology, Vellore, India. Email: rishavagarwal2717@gmail.com

Samridhi Murarka, Computer science and engineering, Vellore Institute of Technology, Vellore, India. Email: samridhi.m98@gmail.com

Debajit Datta, Computer science and engineering, Vellore Institute of Technology, Vellore, India. Email: debajit.datta2000@gmail.com

A. Layers of software architecture:

Presentation layer: User interface is the first layer of software application. It translates works and the outcomes into those which users understands easily.

Application layer: This layer involves in activities like commands processing, calculations evaluation, decisions which are logical. It acts as an intermediate between the other two layers.

Data layer: The lowest layer of the architecture. Database helps in storing and getting all the details. Then these details are reverted back for evaluation to logical layer and then to user.

B. Code dependencies

Dependency is a broad software engineering term used to refer when a piece of software relies on another one. The extent to which one module depends on other is called a dependency. Program X uses Library Y.

Mainly there are two types of code dependencies: Include dependencies (Internal dependencies): The include dependencies include the header files of the program through which we import some main functionalities of the program.

Ex: #include<stdio.h>.

Symbol dependencies (External dependencies): These are the dependencies between activities of a project and outside the project which have to be included in the schedule of a project. They include the dependency between the code files which we use in the software. We show them using dependency graphs. A project consists of many files in which many files will be dependent on the other code files. We can view them as a tree concept in such a way that the subtrees or the children of the previous nodes depend on their parent node which has the main functionality. The code dependencies mentioned show a great impact on architecture recovery techniques. Software architecture recovery techniques help in getting back the original ground truth architecture of the software which was setup by the developers while implementing the software.

II. RELATED WORKS

[1] This paper compared several architecture recovery techniques based on their effectiveness and applicability. This paper introduces the recovery of basic versions of ground-truth architecture using a new module-based technique. This paper also concludes the importance of accuracy of code dependencies used for recovering software architecture. **[2]**

The authors of this paper in their research developed and implemented a scalable and precise tool that could precisely extract code dependencies of software projects written in C/C++.This tool helps in identifying the dependencies which are inconsistent in nature.

This tool helps the developers to perform large-scale refactoring tasks.**[3]** Igor Ivkovic and Nenad Medvidovic in their paper used the ground-truth architecture as the basis for comparative analysis of different software recovery techniques. They have assessed several recovery techniques using different metrics to identify the components and structure of a system architecture. This paper identifies several paths for further exploration in software architecture recovery. **[4]** Ivo Krka and Chris Mattmann in e their paper presented their experiences in recovering the ground -truth architecture of open source systems namely Chromium, Hadoop, ArcStudio and Bash.Their study focuses on the feasibility of obtaining system ground truth architecture for large systems. **[5]** Zhihua Wen and Vassilios Tzerpos in their research paper introduced an effective measure for software clustering algorithms based on MoJo distance. This paper also explains the vibrant features of MoJoFM. MoJoFM is a distance metric for software clustering. The evaluation of the similarity between two different decompositions of the system software is usually performed using this distance metric. **[6]** Mark Shtern and Vassilios Tzerpos in their research paper discuss the challenges faced during the evaluation of the effectiveness of several software clustering algorithms. This paper introduces a novel set of indicators that are used to evaluate the effectiveness. The goal of this research paper is to study the reasons for the discrepancies in MoJoFM. **[7]** This paper discusses new hybrid algorithms used for software clustering such as basicMQ and TurbboMQ. This paper compares the stability and feasibility of the algorithms.

III. FRAMEWORK MODEL

If you are using *Word*, use either the Microsoft Equation Editor or the *MathType* add-on (http://www.mathtype.com) for equations in your paper (Insert | Object | Create New | Microsoft Equation *or* MathType Equation). "Float over text" should *not* be selected.

A. Approach

- Extract the dependencies of the project.
- Give these dependencies as input to the recovery techniques (algorithms).
- Algorithms to be used: ACDC, WCA, LIMBO.
- Get the subsystems of the entire software system through these algorithms as output.
- Compare the results with the results of the k means clustering technique.

B. Extraction and visualization of code dependencies

For extracting and visualizing the code dependencies from the code bases, we're going to use a software called "Softagram". It is mainly used for extracting the code dependencies of mainstream languages such as C, C++, C#, Java, JavaScript, Python and PHP. This software collects metrices like LOC and commit counts to help identifying bottlenecks. It provides UI for displaying the code dependencies. This software has a feature called as "Internal dependency view" which displays how the repositories are dependent on each other.

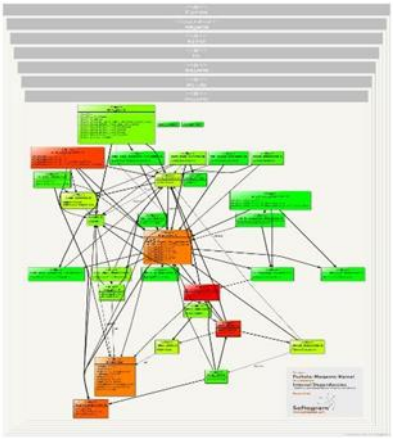

Figure. 1 Code dependency graph of the project "Fuchsia" by Google

C. Figures and Tables

1. K means clustering

It is a clustering technique. It groups modules in a mathematical way by considering the centroid of the modules which we want to cluster. K means can do clustering at central and hierarchical level. Central level-clustering between the different modules. Hierarchical level – clustering within the modules. As shown in the picture at first the data sets or modules are considered to be placed on a 2d plane and two centroids or mean values are placed randomly on the plane. Then a Euclidean line is considered in which the data sets which are on the either side of that line are assumed to be closer to the centroids which are present on those respective sides.

The mean of these data sets is taken and a new centroid will be formed and thus the centroid position changes making it closer to those particular data sets on one side of that Euclidean line. This similar procedure is followed for the data sets on other side of that line. Now again a Euclidean line is considered and the data sets which are on the either side of that line are assumed to be closer to the new centroids which are present on those respective sides. Hence the location of centroid changes again and it becomes closer to those sets. The whole procedure is continued until the number of data sets which were found in last but first iteration and the last iteration are equal. This is the way in which the k means algorithm is used in making clusters.

2. ACDC (Algorithm for Comprehensive Driven clustering)

It's a technique for software clustering and hence it is used in decomposing large software systems into subsystems. It gives clusters which follow commonly occurring patterns while decomposing large software systems. It follows a pattern driven approach. It identifies the interaction between the entities (procedures and variables). It creates clusters with limited number of objects. The grouped clusters help in better understanding of a program.

Some ways through which subsystem patterns can be derived through clustering are: In a source file, set of procedures and variables can be combined to form a cluster. Clustering based on pattern of source files. A set of independent files functioning on a purpose which is similar. Supporting libraries pattern. Set of procedures accessed by majority of subsystems. The ACDC recovery technique maintains the system's decomposition as the system evolves Why K Means does not show better results when compared to ACDC algorithm?

K means algorithm works best with limited data sets(modules). But if the there are many data sets, the whole graph becomes messy. Software systems contain many modules and its dependency graph is too messy. So, it(k-means) will be unable to differentiate between the clusters. Hence k means does not show better results when compared to ACDC algorithm.

3. LIMBO Algorithm

It is a hierarchical clustering algorithm. It gives the data's reduced model on which clustering is performed. A group of objects are summarized in a dcf. Then by following the scalable bottleneck algorithm a dcf tree is built by clustering child dcf's. All the clustered data sets(summary) will be in the leaf nodes dcf's. The intermediate nodes include only those which are emerged by combining those dcf nodes which are children to them.

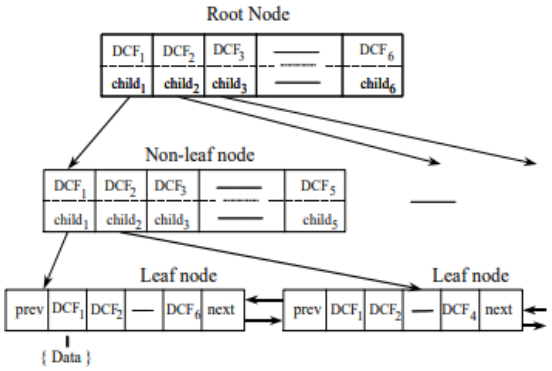

Figure. 2 Working of LIMBO algorithm

4. Weighted Combined Algorithm

Weighted combined algorithm is a hierarchical clustering technique which is based on the code dependencies. This algorithm measures the distance between the cluster of the software entities and then groups them based on their cluster distance. The algorithm proceeds with single cluster which has an associated feature vector. The cluster distance is calculated for all the clusters and the clusters which are similar are merged or grouped together. Here, the user defines the specific number of clusters to be grouped. The process continues until the algorithm reaches the number of clusters specified by the user.

IV. DISCUSSION

MoJoFM: MoJoFM is a distance metric for software clustering. The evaluation of the similarity between two different decompositions of the system software is usually performed using this distance metric. The software resources are basically divided into sets. These sets are further divided into partitions. MoJoFM metric calculates the partition distance of these sets. From the table given below, we can observe that the include dependencies dominantly improve the accuracy of the recovered architecture over symbol dependencies except for ACDC.

Table 1. MoJoFM results for Bash

Algorithm	Include dependencies	Symbol dependencies	Transitive dependencies	Functional dependencies
ACDC	51	56	37	48
WCA-UE	33	23	23	28
LIMBO	33	26	26	21
K-means	58	54	48	46

Architecture-to-Architecture (a2a): a2a measures the distance between the two architectures. Given below are the a2a results of the Bash project. From this table we can conclude that the combination of systems and techniques which use these dependencies rely more on symbol dependencies than on include dependencies. It can be observed that K-means show greatest improvement when it uses symbol dependencies compared to include dependencies. WCA-UE follows K-means in this aspect.

Table 3. a2a results for Bash

Algorithm	Include dependencies	Symbol dependencies	Transitive dependencies	Functional dependencies
ACDC	64	79	79	40
WCA-UE	64	80	80	39
LIMBO	62	78	78	37
K-means	66	83	83	40

Normalized Turbo Modularization Quality: Based on the dependencies, the cohesion and organization clusters quality are measured using the metric called "TurboMQ". Here we can see that the TurboMQ scores of the symbol dependencies are greater than that of include dependencies. This clearly depicts that the symbol dependencies assist software recovery techniques obtain architecture with better cohesion than the include dependencies

Table 3. TurboMQ results for Bash

Algorithm	Include dependencies	Symbol dependencies	Transitive dependencies	Functional dependencies
ACDC	8	21	5	28
WCA-UE	0	6	6	9
LIMBO	7	12	7	6
K-means	0	16	5	13

Comparison of software architecture recovery algorithms with baseline algorithms

Here, we will be comparing ACDC, WCA and LIMBO algorithms with K-means (a simple unsupervised machine learning algorithm). From the table given below, we can conclude that the only algorithm to produce consistently better results than K-means is ACDC. In the above comparison, we have taken all the metrics into consideration.

WCA and LIMBO always gave worst results when compared to K-means.

Table 4: Wilcoxon Signed Rank for Each Algorithm When Compared to K-Means

Metrics	ACDC	WCA-UE	LIMBO
MoJoFM	<.001	<.001	<.001
a2a	.42	.11	<.001
TurboMQ	<.001	<.001	<.001

V. CONCLUSION

The paper explains the basic structure of the software architecture and how its individual components are dependent of different types of dependencies such as include dependencies and symbol dependencies. We also study the different factors which affect the accuracy and efficiency of the recovery techniques. In order to measure the impact of dependencies on software architecture recovery techniques, we have used certain metrics such as MoJoFM and Normalized TurboMQ. This paper compares the efficiency of three different software recovery techniques namely, ACDC (Algorithm for Comprehensive Driven Clustering), LIMBO and WCA (Weighted Combined Algorithm). We have taken K-means as the baseline algorithm for the comparison. Generally, most of the recovery techniques extracted better ground truth architecture when using symbol dependencies compared to include dependencies. There are many other metrics that can be used to compare the efficiency of these recovery techniques which significantly makes some room for more research and exploration. There are some architecture recovery techniques which might perform better than ACDC which also makes some room for further exploration and research. The research can also be done on the software architecture recovered by these techniques by making use of some parameters.

REFERENCES

1. "Comparing software architecture recovery techniques using accurate dependencies" by Thibaud Lutellier, Devin Chollak, Joshua Garcia, Lin Tan, Derek Rayside.
2. "Generating Precise Dependencies for Large Software" by Pei Wang, Jinqiu Yang, Lin Tan, Robert Kroeger, and David Morgenthaler.
3. "A Comparative Analysis of Software Architecture Recovery Techniques" by Igor Ivkovic and Nenad Medvidovic..
4. "Obtaining Ground-Truth Software Architectures" by Ivo Krka and Chris Mattmann.
5. "An effectiveness measure for software clustering algorithms" by Zhihua Wen and Vassilios TzerposJ.
6. "Refining Clustering Evaluation Using Structure Indicators" by Mark Shtern and Vassilios Tzerpos.
7. "Clustering of Software Systems using New Hybrid Algorithms" by Ali Safari Mamaghani and Mohammad Reza Meybodi.

AUTHORS PROFILE

Prathamesh Borhade is a third-year undergraduate pursuing computer science and engineering from Vellore institute of technology. He is a machine learning enthusiast and an aspiring data scientist currently working at Samsung Prism Project. He has also worked at Rapid Circle as a Machine Learning Intern. Email: prathamesh.borhade29@gmail.com

Rajvardhan Deshmukh is currently pursuing his B.Tech in Computer Science and Engineering from Vellore Institute of Technology, Vellore, India. He is a full stack web-developer with an industrial experience in Pune Municipal Corporation. He is also a Data Science and Machine Learning enthusiast and has done several projects based on it during his three years in college. He is the Projects Head of IEEE Computer Society, Vellore and has been actively participating in several Hackathon. He has also conducted a few Hackathons at the university level. Email: rajvardhan1999@gmail.com

Rishav Agarwal is in third-year. He has developed several projects in the platform of Web Developement. Recently, he's been involved in the field of Data Sciences. He has been an active part of several chapters like IEEE-Computer Society and also been performing as part of the VIT Dance Club. Email: rishavagarwal2717@gmail.com

Samridhi Murarka, is currently a third year undergraduate student pursuing Computer Science from Vellore Institute of Technology, Vellore. She is a UI/UX designer. She is an active IEEE Computer Society member and has led the media and design for the community in several events. She has a passion for integration of technology and business. She is currently researching more on Natural Language Processing and has developed an extractive summarizer as a part of a team of 4 members. Email: samridhi.m98@gmail.com

Debajit Datta is currently a third-year undergraduate pursuing Computer Science and Engineering from Vellore Institute of Technology, Vellore. He is an experienced front-end developer with experience of working in the non-profit organization management industry. He has been an active member of the clubs and chapters in the university and has participated is several hackathons at university level. He has been an active member of Developers Student Clubs VIT, a student chapter powered by Google Developers Group and Venturesity VIT, a student chapter. He has industrial exposures from five different industrial internships within the three years of B. Tech. He has also presented a research paper at ic-ETITE'20 conference organized by IEEE and supported by ACM. Email: debajit.datta2000@gmail.com

.