Research On Software Evolution Reconstruction Based on Architecture Recovery

Linhui Zhong, Haitao Ye and Jing Xia
College of Computer and Information Engineering
Jiangxi Normal University
Nanchang, CHINA
{shiningto@jxnu.edu.cn, 609715176@qq.com, 2574509220@qq.com}

Abstract- In this research, a method for recovering software's evolution history based on software architecture reverse was proposed, and the evolutionary binary-free's constructing algorithm as well as the system for evolutionary binary-tree's construction are introduced. At last, we draw a conclusion that the method is effective by regarding three open source systems as experimental objects.

Keywords-Software Architecture; Reverse Engineering; Evolutionary Binary-Tree; Evolutionary Tree;

1. INTRODUCTION

As a significant asset of software, software architecture, at a higher level, is an abstract description of software system [1]. With the continuous development of software system, software architecture's scale has become larger and its structure has become more and more complex. Besides, it's hard to maintain and reuse lots of legacy systems because many software developers merely aim at source code development while ignoring software architecture construction. Therefore, software architecture reverse is particularly important.

At present, existing software architecture reverse technology simply directs at a single version, recovering software architecture from source code, and there is no research on evolution relationship recovery between versions yet. Hence, aiming at this insufficiency, we propose a method based on the architecture reverse technology to recover the evolution relationship between multiple software versions by evolutionary binary tree. In section two, related researches will be briefly introduced. In section three, the concept and the constructing method of evolutionary binary-tree will be presented. In section four, the system design will be introduced. And in section five, the experiments will be written. Finally, it will come to a conclusion.

II. RELATE WORK

Software architecture recovery is the process to recover the high-level information of a system based on low-level descriptions (usually the source code) of the system. For software architecture recovery, various methods were proposed to recover software architecture. For instance, Tzerpos [2] developed the tool ACDC which helps to restore software architecture through analyzing source program by model-driven clustering algorithm. Mitchell [3] analyzed the source code of a system in use of Bunch to restore software architecture with the help of obtaining a class dependency graph by clustering according to the dependency relationship between classes. Cai [4] proposed a software architecture recovery method based on the combination of design rule clustering and other clustering techniques. Wang [5] proposed a method for evolution traceability recovery by combining keyword-based retrieval and heuristic rules was proposed. Garcia [6] presented a machine learning technique to restore an architectural view which contains system components and connectors. Medvidovic [7] put forward a lightweight architecture recovery method. It can not only restore the system architecture, but also re-build the system. Tonella [8] introduced the analysis about medium-sized applications in the light of schema-based architecture recovery environment and obtained useful information of the system software architecture. Sartipi [9,10] demonstrated a high-level design to restore legacy software systems based on user-defined architectural patterns and graph matching techniques. Harris [11] showed a recognizer which can extract the system architecture from the system source code. Antoniol [12] recommended a method of recovering design patterns from Object-Oriented software source code. Jansen [13] presents ADDRA, an approach an architect can use for recovering architectural design decisions after the fact. Li [14] proposed an improved genetic algorithm to achieve the restoration of software architecture. Servant [15] presented Fuzzy History Slicing, a multi-version code-history analysis technique, which has higher accuracy than the existing code-history analysis technique. Aghajani [16] developed a visualization plug-in to depict the history of any chosen file augmented with information mined from the underlying versioning system. Mokni [17] presented an evolution management model that generates evolution plans according to a given architecture change request.

III. THE CONSTRUCTING METHOFDS OF EVOLUTIONARY BINARY-TREE

Actually, the course of software evolution history can be plotted as a tree, i.e. evolutionary tree. In our research, each evolutionary tree was transformed into a binary tree for convenience. Thus, in this section, the definition and constructing methods of evolutionary binary-tree will be introduced.

978 1 53866565 7/18/$31.00 ©2018 IEEE

A. Definition ofEvolutionary Tree and Evolutionary Binary-Tree

Definition1: evolutionary tree: T = (root, F). The word "Root" refers to the root node ofthe tree, and the letter "F" refers to the forest consisting of $m(m\geq 0)$ sub-trees.

Definition2: evolutionary binary-tree: T = (root, L, R). Transformed from evolutionary tree, it is also a binary tree in essence. The letter "L" refers to the left child of the root node and the letter "R" refers to the right child.

1) Evolutionary branch: The research stipulates that the main evolutionary branch starts from its root node, along the left node, to evolutionary branch which consisting of Nn leaf evolution nodes, and the side evolutionary branch starts from the root node, along the right node, to the leaf node.

2) Virtual node: For any node N, the right child N.R, if it exists, is called as a copy of N. It can be formulated as N. R-ClonerN). The copy node N.R is defined as a virtual node retaining the component name of its parent node and its version number generated by the system.

3) Real node: It refers to all non-copy nodes in which the baseline version of the system version corresponding to the component version, the component name, the multi-dimensional attribute array, and the version number generated by the system are stored.

B. Evolutionary Binary-Tree 's Construction

Here is the pseudocode for constructing evolutionary binary-tree

```
Algorithm: Build Evolution Binary Tree
Procedure addTreeNode(Tree tree, TreeNode treeNode)
 If (tree.root = null) Then
   version = addRoot(tree, treeNode);
 Else
   pnode = findSimilaryTreeNode(tree, treeNode);
   similarity = calculateSimilarity(pnode, treeNode);
   If (similarity = 1) Then
     version = addSameNode(pnode, treeNode);
   Else
     Do While pnode != null
     If(similarity>=threshold)And(pnode.LNode=null)Then
         version = addLeftNode(pnode, treeNode);
         Break While;
       Else (similarity >= threshold)
                   And (pnode.LNode!= null) Then
         pnode = findNextSimilaryTreeNode(pnode);
         similarity = calculateSimilarity(pnode, treeNode);
       Else
         Do While pnode.RNode!= null
           pnode = pnode.RNode;
         End While
         version = addRightNode(pnode, treeNode);
         Break While;
       End If
     End While
   End If
 EndIf
 Return version
End Procedure
```

C. *Similarity Measure*

It is necessary to make a similarity measure to compare the attribute arrays between the new node to be insertion and the existing node when constructing an evolutionary binary-tree. Thus, the research measures the similarity by the following cosine formula:

$$Sim(S1, S2) = \frac{\overrightarrow{S1} * \overrightarrow{S2}}{\|S1\| * \|S2\|} \quad \frac{\sum_{i=1}^{n} a_i * b_i}{\sqrt{\sum_{i=1}^{n}(a_i)^2} * \sum_{i=1}^{n}(b_i)^2}$$

"SI" represents the attribute vectors of version 1 and "S2" refers to that of version 2, and the similarity value between the two versions is in the range of (0,1]. The closer the value is to 1, the more similar the two versions are, and vice versa.

IV. SYSTEM DESIGN

In this section, our evolutionary binary-tree's constructing system will be introduced. The system is mainly divided into five modules: source code reading module, architecture reverse module, attribute measurement module, evolutionary binary-tree's construction module, and similarity comparison between evolutionary binary- trees module.

A. Source Code Reading Module

This module is used to read the source code of a system to analyze software evolutionary binary-tree's constructing system so that the source code can be analyzed by other modules later.

B. Architecture Reverse Module

We use the tool Bunch to restore software architecture view in the basis of the dependency relationship between classes of source code. According to the research, it is shown that Bunch is the best tool for architecture reverse [18].

The module IS used to recursively generate the corresponding atomic components and composite components of the system through combining with the class dependency diagram obtained in previous step. A composite component corresponds to a certain system version, and each composite component contains many atomic components.

C. *Attribute Measurement Module*

This module is mainly to measure various attributes of the system to be analyzed, including the information about atomic components, the number of atomic components, the size of atomic components, and the size of the software architecture (the research uses graph edit distance algorithms to quantitatively size the software architecture.) relating to the software architecture, and other data relating to the system source code, such as the number of valid code lines, and the number of java files. What's more, it also has the ability to not only measure each atomic component's three-dimensional attributes (the number of classes, class files and the sum of classes' sizes all contained in atomic component) of composite component but save them for the construction of the atomic component's evolutionary binary-tree.

D. *Evolutionary Binary-Tree 's Construction Module*

This module is the core of the evolutionary binary-tree. It uses the three-dimensional attributes measured before to construct an evolutionary binary-tree of each atomic component. We combine the five attributes measured before to form a whole attribute vector, then construct an evolutionary binary-tree by the construction algorithm proposed in section three. Furthermore, our system allows to select attributes for users to construct various evolutionary trees.

E. Similarity Comparison Between Evolutionary Binary-Trees Module

This module primarily aims at comparing the similarities between evolutionary binary-tree constructed by us and real tree. We calculate the similarity of two binary-trees by a tree edit distance algorithm [19]. The smaller the tree editing distance is, the more similar the evolutionary binary-tree is to the real evolutionary binary-tree, and vice versa. Besides, we can also calculate the similarity between the evolutionary binary-trees (formed by different combinations of various attributes) and the real ones.

Next, we'll briefly demonstrate the evolutionary binary-tree's construction module in our system. We analyzed 12 historical versions of Openjpa. Fig.2 below shows a composite component's evolutionary binary-tree derived from reverse construction by utilizing the attribute combination which consists of the number of atomic components, the size of atomic components, and the size of the software architecture.

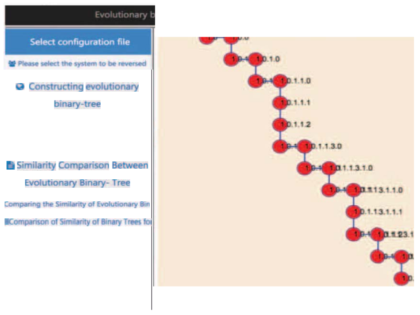

Evolutionary b
Select configuration file
Please select the system to be reversed
Constructing evolutionary binary-tree
Similarity Comparison Between Evolutionary Binary- Tree
Comparing the Similarity of Evolutionary Bin
Comparison of Similarity of Binary Trees fo
1.0.1.0
1.0.1.1.0
1.0.1.1.1
1.0.1.1.2
1.0.1.1.3.0
1.0.1.1.3.1.0
1.0.1.1.3.1.1.0
1.0.1.1.3.1.1.1

Fig1. Evolutionary binary-tree's construction module

V. EXPERIMENTS

In this section, we will demonstrate the experiments performed by the methods mentioned above. We selected the source codes of three open source systems-Cassandra, Hbase, and Hive as experimental data. After analyzing the three systems separately, we used systems' attributes to reverse each evolutionary history of a certain system according to its five attributes. Firstly, we measured the source codes of three systems, then we generated different evolutionary binary-trees with the help of various combinations of attributes. Here are the experimental results:

A. Cassandra's experimental results

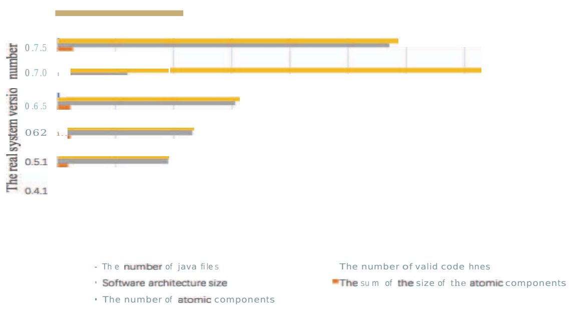


Fig2. Cassandra's Multi-version attribute column garph

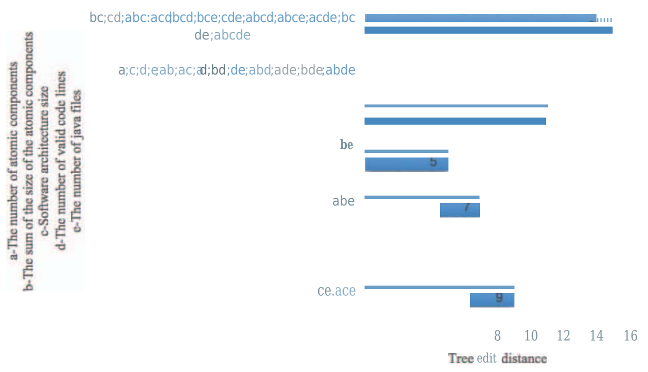


Fig3. Cassandra's tree editing distance of various attribute combination

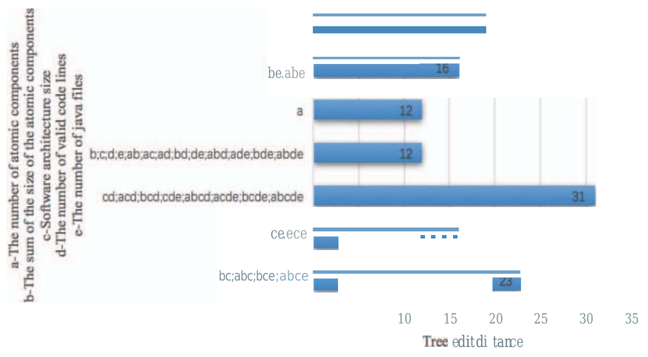
In Fig. 2, we show the attribute values of different versions of the system. In Fig. 3, we plot the tree edit distance between the evolutionary binary tree and the real one generated by combining different attributes. The letters "a", "b", "c", "d", "e" respectively represent the number of atomic components, the sum of the size of the atomic components, software architecture size, the number of valid code lines, and the number of java files. And the semicolon ";" is used to separate different attribute combinations.

According to Fig. 3, it's easy to see that when the attribute combinations are a;c;d;e;ab;ac;ad;bd;de;abd;ade;bde;abde, and the minimum tree edit distance is 3. In other words, this evolutionary binary-tree generated from these attribute combinations is the most consistent one with the real tree.

B. Hbase's experimental results

The real system version number
The number of java files
Software architecture size
The sum of the size of the atomic components
The number of atomc components

Fig4. Hbase's Multi-version attribute column garph



FigS. Hbase's tree editing distance of various attribute combination

According to Fig. 5, it's easy to see that when the attribute groups are a;b;c;d;e;ab;ac;ad;bd;de;abd;ade;bde;abde, the evolutionary binary-tree constructed by the system is the closest one to the real evolutionary binary-tree, and the tree edit distance is 12.

C. Hive's experimental results

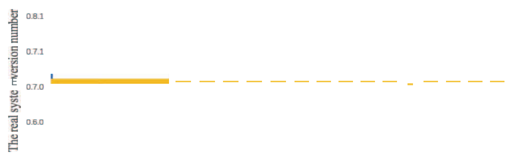


Fig6. Hive 's Multi-version attribute column garph

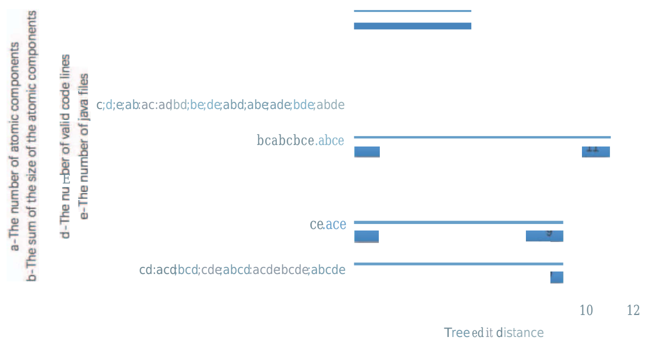


Fig7. Hive 's tree editing distance of various attribute combination

According to Fig. 7, it is shown that when the attribute group is b, the tree edit distance between the evolutionary binary-tree and the real one constructed by the system is 2.

VI. CONCLUSION

Through those experiments, it can be found that the evolutionary binary-tree's constructing method proposed in this research is very effective, because this method can reversely restore the evolution history between versions of legacy system.

ACKNOWLEDGMENT

The article is sponsored by the National Natural Science Foundation project(61462040, 61662032, 61262015,61762049), Jiangxi Provincial Natural Science Foundation Project(20142BAB207027, 20171BAB202013) and Jiangxi Province Education Department for Science and Technology Project(GJJI70207).

REFERENCES

[1] Garlan D. Software architecture: a roadmap[C]//Proceedings of the Conference on the Future of Software Engineering. ACM, 2000: 91-10 1.

[2] V. Tzerpos and R. C. Holt, "ACDC: An algorithm for comprehensiondriven clustering," in Proceedings of the Seventh Working Conference on Reverse Engineering, ser. WCRE '00. Washington, DC, USA: IEEE Computer Society, Nov. 2000, p. 258. [Online]. Available: http ://d1.acm.org/citation.cfm?id=832307.837118

[3] Mitchell B S, Mancoridis S. On the automatic modularization of software systems using the bunch tool[1]. IEEE Transactions on Software Engineering, 2006, 32(3): 193-208.

[4] Cai Y, Wang H, Wong S, et a1. Leveraging design rules to improve software architecture recovery[C]/lProceedings of the 9th international ACM Sigsoft conference on Quality of software architectures. ACM, 2013: 133-142.

[5] WANG Jin-shui, AI Wei, PENG Xin, et a1. Recovering Traceability Links among Multi-level Software Evolution Information [J]. Computer Science, 2012, 39(7): 135-139.

[6] Garcia J, Popescu D, Mattmann C, et a1. Enhancing architectural recovery using concerns[C]/lProceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering. IEEE Computer Society, 2011: 552-555.

[7] Medvidovic N, Jakobac V. Using software evolution to focus architectural recovery[J]. Automated Software Engineering, 2006, 13(2): 225-256.

[8] Tonella P, Fiutem R, Antoniol G, et a1. Augmenting pattern-based architectural recovery with flow analysis: Mosaic-a case studyl Cl/zkeverse Engineering, 1996., Proceedings of the Third Working Conference on. IEEE, 1996: 198-207.

[9] Sartipi K. Software architecture recovery based on pattern matchingj'C'[z/Software Maintenance, 2003. ICSM 2003. Proceedings. International Conference on. IEEE, 2003: 293-296.

[10] Sartipi K, Kontogiannis K, Mavaddat F. Architectural design recovery using data mining techniques[C]//Software Maintenance and Reengineering, 2000. Proceedings of the Fourth European. IEEE, 2000: 129-139.

[11] Harris D R, Reubenstein H B, Yeh A S. Recognizers for extracting architectural features from source code[C]/lReverse Engineering, 1995., Proceedings of 2nd Working Conference on. IEEE, 1995: 252-261.

[12] Antoniol G, Casazza G, Di Penta M, et a1. Object-oriented design patterns recovery[J]. Journal of Systems and Software, 2001, 59(2): 181-196.

[13] Jansen A, Bosch J, Avgeriou P. Documenting after the fact: Recovering architectural design decisions[J]. Journal of Systems and Software, 2008, 81(4): 536-557.

[14] Li QS, Chen P. Implementing architecture recovery by using improved genetic algorithm. Journal of Software, 2003,14(7):1221~1228.

[15] Servant, Francisco, and James A. Jones. "Fuzzy fine-grained code-history analysis." Proceedings of the 39th International Conference on Software Engineering. IEEE Press, 2017.

[16] Aghajani, E., Mocci, A., Bavota, G., & Lanza, M. (2017, May). The code time machine. In *Program Comprehension (ICPC), 2017 IEEE/ACM 25th International Conference on* (pp. 356-359). IEEE.

[17] Mokni, A., Urtado, C., Vauttier, S., Huchard, M., & Zhang, H. Y. (2016). A formal approach for managing component-based architecture evolution. *Science of Computer Programming*, 127, 24-49.

[18] Wu J, Hassan A E, Holt R C. Comparison of clustering algorithms in the context of software evolutionjCl/zboftware Maintenance, 2005. ICSM'05. Proceedings of the 21st IEEE International Conference on. IEEE, 2005: 525-535.

[19] Zhang K, Shasha D. Simple fast algorithms for the editing distance between trees and related problems[J]. SIAMjournal on computing, 1989, 18(6): 1245-1262.