PROCEEDINGS OF SCIENCE

Understanding of the Component-Based Software Evolution by Using Similarity Measurement

Xiaozheng Zhu[1]
Jiangxi Normal University, College of Computer and Information Engineering
Nanchang, Jiangxi, 330022, China
*E-mail:*381506605@qq.com

Linhui Zhong[23]
Jiangxi Normal University, College of Computer and Information Engineering
Nanchang, Jiangxi, 330022, China
E-mail: shiningto@jxnu.edu.cn

Hongyan Zong
Jiangxi Normal University, College of Computer and Information Engineering
Nanchang, Jiangxi, 330022, China
E-mail: 736994262@qq.com

Changyuan Hou
Jiangxi Normal University, College of Computer and Information Engineering
Nanchang, Jiangxi,330022, China
*E-mail:*1137470916@qq.com

Nengwei Zhang
Jiangxi Normal University, College of Computer and Information Engineering
Nanchang, Jiangxi, 330022, China
E-mail: 237438508@qq.com

The traditional methods of understanding software evolution by using measurement primarily focus on file, directory or project, and take measurement on software attributes (such as software complexity, modularity and software reusability. etc.), which lack the ability of measuring software evolution at a higher level. In this position paper, we propose an approach to measure the evolution similarity between the component-based software based on the attribute change, and focus on designing an interpolation algorithm to deal with the situation of different versions when taking measurement. Experimental results are given to show the utility of the algorithm.

CENet2015
12-13 September 2015
Shanghai, China

[1]Speaker
[2]Corresponding Author
[3]The article is sponsored by the National Natural Science Foundation Project (No.61262015, No.61462040), Jiangxi Provincial Natural Science Foundation Project (20142BAB207027,20142BAB207011), Jiangxi science and technology support project (20142BBE50028).

PoS(CENet2015)077

1. Introduction

The software evolution is a process of developing software initially and then repeatedly updating it for various reasons. Currently, more and more attentions have been paid to the software evolution. The workshop about mining software repository focuses on the theme about how to mine useful information from the software repository to the improve the software quality and reliability.

In order to measure the change of software, two problems have to be solved as follows: (1) how to obtain the software historical data or the software evolution information. Software evolution information is mainly stored in CASE tools such as the software configuration management system and the error reporting system, which use the file or project as the basic unit to record software changes; however, with the popularization of the component-based software development, the concept of component and software architecture cannot directly be mapped into the software configuration management system and it will become more difficult in dealing with the component-based software evolution information. (2) Lack the technology of measuring the software evolution information. At present, the measurement of software evolution mainly aims at change of the size or structure of source code. With the increasing complexity of software system, it becomes much more difficult in understanding and measuring the software evolution; therefore, it's necessary to put forward a new measurement technology for component-based software evolution.

In order to solve the above problems, we propose an evolution similarity measurement based on the traditional measurement and focus on the virtual version insert algorithm for the component-based software because the evolution similarity measurement might face the problem of having different versions. This work is organized as follows; related work is discussed in Section 2. In Section 3, we'll describe the evolution similarity measurement and the virtual version insert algorithm, finally the conclusion is given.

2. Related Works

2.1 Software Evolution Measurement

The traditional software measurement mainly aims at measuring a single software's attributes (e.g. software complexity, modularity and reusability degree); however, the software evolution measurement is different from the measurement for a single software system to examine multiple versions of software system[1][2].

At early times, the software evolution measurement was focusing on the number of change of software system code and module, etc. For example, Lehman discovered the Laws of Software Evolution, on the basis of studying the history of operating system IBM360/370[3], found the characteristic of the linear tendency of software evolution by measuring the change number of module. Later, Roble studied the kernel subsystems to measure the change of SLOC (Source Lines of Code) and made a conclusion that the software evolution had the characteristic of hybrids (linear/super-linear) increasing [4].

In recent years, researchers have been trying to deeply make insight on software evolution measure from model, structure change or defects of software. For example, Pamela proposed a method to predict the cost of software development and maintenance by measuring the topology of Graph, including the diagrams based on the source code (such as call graph, module collaboration diagrams.etc.) and the collaboration diagrams among developers based on bug and change requests[5]. Christoph took OWL (Ontology Web Language) as data exchange format for software repository, based on which he analyzed the software system evolution by designing a query engine iSPARQL and a query language SPARQL extended the function of RDF, which can be applied to the software evolution visualization, software measure and code bad smells detection[6]. Alexander Chatzigeorgiou proposed to adopt DEA (Data Envelopment Analysis) as a means of providing a unified view of selected design metrics. DEA aimed at assessing the overall trend of quality during the evolution of software systems and it enabled the perception of

global trends in qualitative characteristics [7]. Gregorio Robles proposed to move from the physical towards a level that includes semantic information by using functions or methods for measuring the evolution of a software system and they pointed out that use of functions-based metrics may has many advantages over the use of files or lines of code [8].

2.2 Similarity Measure

The similarity measurement is the distance between various data points[9], which are used in measuring the similarity between sets based on the intersection of two sets. It's known that as a function that computes similarity degree between a pair of text objects in the area of information retrieval, it can also be used in the area of computer science. For example, Hierarchical Similarity Measurement Model (HSM) of program's execution is proposed, which avoids having to explicitly form an equation by work like a Black-box model[10]. It uses a similarity value to compute the fitness function and supports primitive, abstract and complex data types. The similarity measurement technology in information retrieval (IR) is also used to reveal the basic connections between features and computational units in the source code [11].

3. Evolution Similarity Measurement of Component-Based Software

Different from other similarity measurements, the evolution similarity measurement of component-based software based on property should consider two aspects (used in 3.1), that is, 1) the software architecture remains unchanged with the component changing. In that situation, the system change can be considered as the overall change on all components; 2) the software architecture is changed, which means the change can be calculated by using the editing distance as the minimum number with editing operation (insert, delete, replace) in the transformations between the tree or graph [12].

3.1 Evolution Similarity Measurement

The evolution similarity measurement refers to the measurement of its similarity of change with the given attributes (e.g. the number of files, etc.) during a period. Generally, the software architecture can be considered as a composite component; therefore, in this article, we don't make a distinction between software architecture and component in the evolution similarity measurement except selection of the measurement attributes, that is, when the software architecture evolution similarity is measured, we not only consider the file attributes but also the structure attributes. The formula of the evolution similarity measure is as follows:

$$\mathrm{Sim}_p(C1,\ C2)=\frac{\overrightarrow{C1}*\overrightarrow{C2}}{\|C1\|*\|C2\|}=\frac{\sum_{i=1}^{n} a_i*b_i}{\sqrt{\sum_{i=1}^{n}(a_i)^2}*\sqrt{\sum_{i=1}^{n}(b_i)^2}} \tag{3.1}$$

Where, $\overrightarrow{C1}$ and $\overrightarrow{C2}$ are separately expressed as the change vector about the attribute *P* of Component C1 and C2; that is, the attribute change vectors of Component C1 and Component C2 is corresponding to $< a_1, a_2, a_3, \ldots, a_i, \ldots, a_{n-1}, a_n >$ and $< b_1, b_2, b_3, \ldots, b_i, \ldots, b_{n-1}, b_n >$. Obviously, the value of the evolution similarity measurement is ranging from 0 to 1, the higher the number is, the higher evolution similarity will be, or vice verse.

3.2 Interpolation Algorithm on Virtual Version

The evolution similarity measurement formula requires that the change vector of Component C1 should have the same length as Component C2; however, when the versions of two components are extracted during a specified period, we cannot always ensure that the number of version for each component is the same. For example, Component C1 may choose versions as $VERS_{C1} = \{V_1, V_2, V_3, V_4, V_5\}$, and the versions of Component 2 is $VERS_{C2} = \{V_1, V_2, V_3\}$, which lead to different dimensions of change vector in the evolution similarity

measurement. In this case, the dimension reduction or the dimension raising method can be used to make change vector to have the same length. In this paper, the dimension raising method is adopted, in other words, a number of virtual versions are inserted in the low dimensional vector. For example, $VERS_{C2} = \{V_1, V_2, V_3\}$ may be turned into $VERS'_{C2} = \{V_1, V_1^*, V_2, V_2^*, V_3\}$, where V_1^*And V_2^* are the virtual versions, whose contents are respectively the same as that of V_1 and V_2. The algorithm of translation from $VERS_{C2}$ to $VERS'_{C2}$ should meet the following basic principles, and the detail of algorithm is also given.

■ Basic principles that the algorithm should meet.

1) After insertion, the sum of the time gap in the corresponding position among different components should be as small as possible.
2) The insertion of virtual versions should keep the component's evolution trend as possible

■ Algorithm: virtual version insert

Firstly, we define the data structure of *componentversion* as follows:

```
Public class Component version {
    String component name; // name of component
Array List property value = new Array List ();//attributes value of component
    Date create time; //the deadline of component; ……. }
```

- **Input**: the *componentlist* vector and *componentlist0* vector, which have different lengths and contain some data, such as the component name and the component attributes etc.
- **Output**: the *midlist* stores the two vectors after inserting virtual version
- **Steps**:
  - Step 1: it takes the submission time of the version of *componentlist* and *componentlist0*, read into *alist* and *blist* respectively, and the *alist* extract the high dimensional vector data, the other extract the low;
  - Step 2: try to choose a position to insert virtual version in the *blist*. At first, we should add 1 on the length of array *blist*, and try to insert a virtual version from the end position to the top position. The detail is shown as follows.
    1. It will set the deadline of the virtual inserted version as blist.set (j, (blist.get(j-1)*2-blist.get(j-2))), if the position inserted is *j= blist.size()-1*.Then it sums the time gap of the data in array *alis*t and *blist* before number j.

```
int sum = 0;
for (int k = 0; k <= i; k++) {
int sub = alist.get(k) - blist.get(k);
sum = sub + sum;}
```

    2. It will move the location of array *blist* behind the number *j* back one position, *blist. set (j+1, blist.get(j))*, if the inserted location is *j >=0 && j < blist. size()-1*; if the inserted position is *j==0*, it will set the attribute *changevalue* of the virtual version as *blist.set (j,((blist.get (j+1))*2-blist.get(j+2)));* otherwise, set it as *blist.set(j, (blist.get(j-1) + blist.get (j+1))/2)*. Then it sums the time gap of the data of two array *alist* and *blist* before number *j.*
  - Step 3: select the position of virtual version. According to arithmetic bubble, we select what can make the absolute value of sum to the minimum insertion into the position; then save the *blist*'s data into midlist.

```
intmid = MAX; //a max value
        if (Math.abs(mid) > Math.abs(sum)) {
            mid = sum; location = j + 1;
            for (int k1 = 0; k1 < asize; k1++) {
    midlist.set(k1, blist.get(k1));
}// select the right location data into the middle of the array}
```

  - Step 4: Start a looping execution with Step 2 and Step 3 from *i=blist.size() to i<alist.size().*
  - Step 5: return the data in variable *midlist*

3.3 Experiment

We experiment the algorithm in three open source projects, that is TreeView, JavaGeom and PasswordSafeSWT. TreeView is a simple program for displaying phylogenies on Apple Macintosh and Windows PCs. Some more details in website; The aim of JavaGeom is to provide methods to easily perform geometric computa -tions. The last is a Java version of the PasswordSafe password management utility, which allows people to manage multiple passwords easily and securely.

As to the specified time period (from September 1, 2009 to March 1, 2014), we mark three extracted version components respectively as $VERS_{TreeView}=\{V_1, V_2, V_3\}$, $VERS_{JavaGeom}=\{V_1,V_2,V_3,V_4,V_5\}$ and $VERS_{PasswordSafeSWT}=\{V_1,V_2,V_3, V_4\}$ from the software configuration management system. The history data is shown in table 1 and 2. The horizontal dimension of the table header indicates the version sequence of component, it's the deadline of each of the components in Table1, and the number of files in Table 2. There is a process to make the dimension of TreeView and PasswordSafeSWT equal before we process the evolution similarity measurement with TreeView and JavaGeom, PasswordSafeSWT and JavaGeom.

	1	2	3	4	5
JavaGeom	2009-09-06	2010-11-06	2011-12-04	2012-07-15	2014-02-23
TreeView	2009-09-08	2010-06-08	2014-01-28		
PasswordSafeSWT	2009-02-15	2009-11-12	2010-12-21	2011-03-25	

Table 1: Deadline of Version of Components: TreeView, JavaGeom and PasswordSafeSWT

	1	2	3	4	5
JavaGeom	144	151	165	162	163
TreeView	488	352	357		
PasswordSafeSWT	110	130	145	146	

Table 2: Number of File of Components: TreeView, JavaGeom and PasswordSafeSWT

Fig.1 shows the change for each version of Component *TreeView* and *Password -SafeSWT* (the horizontal dimension represents the version sequence. The vertical one represents the attribute value) by using our algorithm to improve the dimen-sion of component version list $VERS_{TreeView}$ and $VERS_{PasswordSafeSWT}$. $VERS_{TreeView}$ inserts the virtual version V_3^*, V_4^* between Number 3 and 4 respectively, $VERS\text{-}_{PasswordSafeSWT}$ inserts the virtual version V_4^* into number 4, and it gets lists $VER_{Tree\text{-}View}=\{V_1,V_2,V_3^*,V_4^*,V_3\}$ and $VER_{PasswordSafeSWT}=\{V_1,V_2,V_3,V_3^*,V_4\}$ after insertion.

	1	2	3	4	5
JavaGeom	144	151	165	162	163
TreeView	488	352	**216**	**80**	357
PasswordSafeSWT	110	130	145	**160**	146

Table 3: Number of File of Components: TreeView, JavaGeom and PasswordSafeSWT

Table 3 illustrates the version change historical data of component *TreeView* and component *PasswordSafeSWT* by inserting a virtual version (Table 3 shows the virtual version number in bold). According to Table 3, the corresponding attributes change vector of components *JavaGeom*, *TreeView* and *PasswordSafe SWT* are $AC_{JavaGeom}=<7,14,3,1>$, $AC_{TreeView}=<136,136,136,277>$, $AC_{PasswordSafeSWT}=<20,15,15,14>$. According to Formula (1), the evolution similarity degree of Components *JavaGeom* and *TreeView* is 0. 6098 in special time, and the evolution similarity degree of components *JavaGeom* and *PasswordSafeSWT* is 0.7919. It can be seen that the evolution similarity with *JavaGeom* and *PasswordSafeSWT* is much higher than others from the change of file.

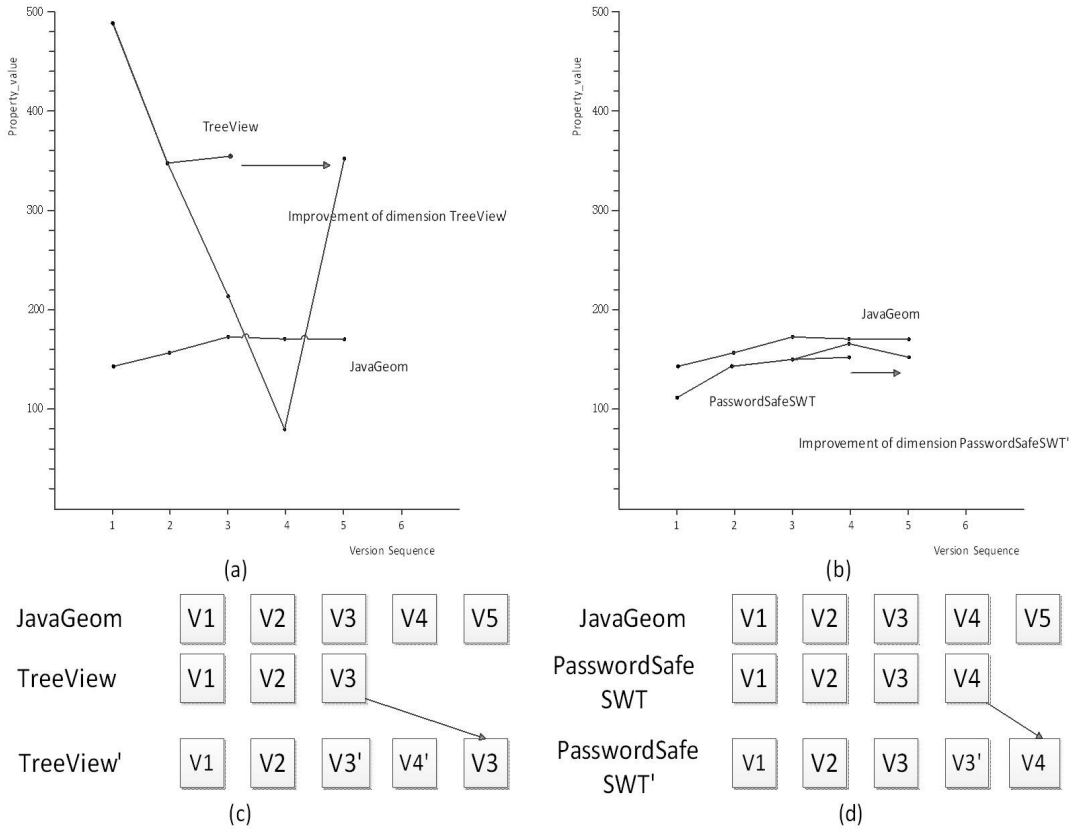

Figure 1: The Same Dimension Processing of Components Input Vector

4. Conclusion

It has been widely accepted that software evolution information can contribute to the development of component-based software.; however, with the increasing amount and complexity of software system, it's more difficult to understand and measure the software evolution. Therefore, in this paper, we take the evolution information of component-based software as research target and component as the base unit of software evolution measurement, propose a method to compare component-based software based on the evolution similarity measurement and the corresponding virtual version insertion algorithm. What should be pointed out is that the current work is experimented at small component-based software system. In the future, we will improve the efficiency of our system and provide automatic processing ability to deal with large component-based software system.

References

[1] P. Louridas, D. Spinellis, and V. Vlachos. *Power laws in software.* ACM Transactions on Software Engineering and Methodology,USA vol.18, no.1, pp.1-26(2008)

[2] Y. Ma, K. He, J. Liu. *Network Motifs in Object-Oriented Software Systems*. Dynamics of Continuous, Discrete & Impulsive Systems, 14(S6): pp.166-172(2007)

[3] M. M. Lehman, J. F. Ramil, P. D. Wernick, D. Perry. *Metrics and Laws of Software Evolution-the Nineties View*, Proceedings of the Fourth International Software Metrics Symposium,IEEE Computer Society.Los Alamitos, California.USA, pp.20-33(1997)

[4] G. Robles, Juan. Jose. Amor. *Evolution and Growth in Large Libre Software Project*, Proceedings of the International Workshop on Principles of Software Evolution,IEEE Computer Society,San Diego,CA. pp.165-174(2005)

[5] P. Bhattacharya, M. Iliofotou, I. Neamtiu, M. *Faloutsos. Graph-Based Analysis and Prediction for Software Evolution*, International Conference on Software Engineering–ICSE,IEEE Computer Society,Pittsburgh, Pennsylvania. pp.419-429(2012)

[6] C. Kiefer, A. Bernstein. *Mining Software Repositories with iSPARQL and a Software Evolution Ontology*, Fourth International Workshop on Mining Software Repositories (MSR'07).IEEE Computer Society,MN,USA.pp. 10-18(2007)

[7] A. Chatzigeorgiou, and E. Stiakakis, *Combining metrics for software evolution assessment by means of Data Envelopment Analysis*, J. Softw.: Evol. and Proc. Journal of Software Evolution & Process, WILEY.USA. 25:pp. 303-324(2013)

[8] G. Robles, and I. Herraiz, *Modification and Developer Metrics at the Function Level*:Metrics for the Study of the Evolution of a Software Project, WETSoM 2012(IEEE), Zurich, Switzerland,pp.49-55(2012)

[9] A. K. Patidar , J. Agrawal, and N. Mishra, *Analysis of Different Similarity Measure Functions and their Impacts on Shared Nearest Neighbor Clustering Approach*, International Journal of Computer Applications, 40(16), pp.1-5(2012)

[10] A. Reungsinkonkarn, *Hierarchical Similarity Measurement Model of Program Execution*, 4th IEEE International Conference on Software Engineering and Service Science, IEEE Computer Society,Milan, Italy.pp.255 – 261(2013)

[11] B. Dit, M. Revelle, and D. Poshyvanyk , *Integrating Information Retrieval, Execution and Link Analysis Algorithms to Improve Feature Location in Software*, Empirical Software Engineering (EMSE), 18(2), pp. 277-309(2013)

[12] T. Sager, A. Bernstein, M. Pinzger, and C.Kiefer. *Detecting Similar Java Classes Using Tree Algorithms*.2006, In Proc. of the 2006 Int. Ws. On Mining Software Repositories (MSR'06)IEEE Computer Society, New York,NY.pp.65-71(2006)