International Journal of Security and Its Applications
Vol. 10, No. 3 (2016), pp.21-30
http://dx.doi.org/10.14257/ijsia.2016.10.3.03

Extracting Clone Genealogies for Tracking Code Clone Changes

Chun-Hui Wang, Ying Tu, Li-Ping Zhang, Dong-Sheng Liu

Computer & Information Engineering College, Inner Mongolia Normal University Huhhot China
ciecwch@imnu.edu.cn

Abstract

Software system's clones are usually two aspects influence on software maintenance and management. One is some clones are effective and can reuse. The other is some clones are unsafe and need revise or reconfiguration. The reason is that the changes of code clones are different. How to determine the clones' attribute of effective or unsafe, it need to track clone changes in the evolution versions of a software system. We firstly find the clones and clone groups in multiple versions of a software system using a clone detector FCD, and construct the mapping of every adjacent version basing on the similarity of code clones, then extract clone genealogies in the software system. The clone genealogies' results are efficient and can help us analysis the code clone changes and get the attribute about effective and unsafe.

Keywords*: clone genealogies, code clone changes, clone mapping, clone evolution patters*

1. Introduction

Software Systems often contain numerous code clones that match each other with varying degree of exactness [1]. These clones are often the result of copy-paste activities. Such activities are very common in systems development, because the developers usually choice copy the code which the function or semantic is similar, and past the clones directly or make some modifies to make sure it can use. This way is convenient and low cost to reuse existing code by copy and paste. However, the modifications of clone codes have impacts on software maintenance and management. Such as the identifiers would be changed or the clone codes maybe need some adds、deletes or changes to suit the new function. If these changes are not consistency among the clone segments, it would cause some errors, bugs or defects. So code clone will increase extra maintenance cost. To help developer maintenance these clones and help them reuse, tracking these code clones' changes in software evolution will let them know which and where changes patterns arise and how to cut down the risk of using clones.

To finds the clone, researchers have developed many approaches and tools. Such as CCFinder[2], NiCad[3], Simian[4]. However, current clone detection tool provide large of information about the clone codes and clone groups (consists of code clones that are clones of each other). The information is too much for users and can't let them know how to use. In addition, the information is static to study clone and can't dynamic tell the user about the clone changes across multiple versions in software systems. For these reasons, some researchers investigated the evolution of code clones throughout a system's history to find the changes of clones and to comprehend and manage the system's clones [5].

Clone genealogy is an efficient method to study code clone evolution across multiple versions. According to the clone fragments of a clone groups to next version's changes, some researchers [6-7] define some clone evolution patters. A clone genealogy describes how the code fragments of a clone group propagate (changes) through versions during the evolution of the subject system [6]. Clone genealogy can help study clone evolution

ISSN: 1738-9976 IJSIA

structurally and semantically rather than quantitatively. Tracking the genealogy, we can get the information in the following:

- Associates clone groups have originated from the same ancestor clone group
- Retrospectively how clones are modified
- To help find that long-live clones and volatile clones.
- To help give some suggestion about reconfiguration to volatile clones or reuse to stable clones.

In this paper, we track the clone change by identifying clone evolution patterns and extracting clone genealogy. To distinguish the different changes of code clone in clone group, we partly use the evolution patterns described by Kim *et al* [6] but more than that. At first, we get the clones and clone groups' data from the clone detector. Then to every two adjacent versions construct the relationship of mapping. At last, according the code clone's changes in clone group for two adjacent versions, we can identify the clone evolution patterns.

The rest of this paper is organized as follows. Section 2 describes related work about clone evolution patterns and clone genealogy. Section 3 gives our approach about how to identify clone evolution patterns and how to extract clone genealogy. Section 4 describes correlation analysis results about tracking the clone changes using clone genealogy. Section 5 describes our study results. Section 6 discusses and summarizes our work.

2. Terminology and Related Work

A clone pair relates two fragments that are similar to each other and are contained in the same version of the system. The degree of similarity based on text and functionalities. The text similarity is often the result of copying a code fragment and then pasting to another location. It included four kinds of similar. Table 1 shows the type and the details.

Table 1. The Similar Type of a Clone Pair

Type	describe	Similarity degree
Type-1	The token sequences of both fragments are identical disregarding whitespace and comments.	Exact clones
Type-2	Structurally/syntactically identical fragments except for variations in identifiers, literals, types, layout and comments.	Exact clones
Type-3	Copied fragments with further modifications. Statements can be changed, added or removed in addition to variations in identifiers, literals, types, layout and comments.	Near-Miss clones
Type-4	it is two or more code fragments that perform the same computation but implemented through different syntactic variants	Functional clones

Some researchers have found that these different types make different influence to clone managements. Type-1 and Type-2 is exact clones, the proportion is more than Type-3 and Type-4. Type-4 is very less than other types and the detecting is more difficult. Our research focus on the Type-1 and Type-2, the reason is that the proportion is large and the method is can spread to other types. A clone group consists of codes that every two clones are a clone pair each other in a software version. To two adjacent versions, the clone group is maybe make changes. Some researchers analyze these changes and define the clone evolution patterns.

2.1 Code Clone Change and Clone Evolution Patterns

To a software system's evolution across multiple versions, some clones never change during evolution, some clones disappear in short amount of time, or some clones undergo repetitive similar edits over their long lifetime. Göde and Koschke[8] demonstrate that less than half of the code clones may change during their life cycle.

Kim *et al*. [6] define six evolution patterns that describe mostly changes to a clone class. OG is an old clone group in the k-1th version, and NG is a new clone group in the kth version. An evolution pattern is a cloning relationship between NG and OG. There are six kinds of evolution patterns.

- Same: all code snippets in NG did not change from OG;
- Add: at least one code snippet in NG is newly added.
- Subtract: at least one code snippet in OG does not appear in NG.
- Consistent change: all code snippets in OG have changed consistently.
- Inconsistent change: at least one code snippet in OG changed inconsistently.
- Shift: at least one code snippet in NG partially overlaps with at least one code snippet in OG.

2.2 Clone Genealogy

Kim *et al*.[6] present the clone genealogy. A clone's genealogy associates related clone groups that have originated from the same ancestor clone group. In addition, the genealogy contains information about how each element in a group of clones has changed with respect to other elements in the same group. Kim *et al*. develop a clone genealogy extractor CGE to get a software systems clone genealogy. Using CGE, they research the relationship between clone lived time and refactoring. Their study discovers that some types of clones that refactoring would not help, CGE can help clone maintenance using clone genealogy information.

Saha *et al*. [7] extended the study by Kim, their method is fit for Type 1-3 clone types. They develop a prototype tool gCad and evaluation of clone genealogies in 17 open source systems cover four different programming languages. Their findings is same with Kim that clones may not always be harmful in software maintenance, and they insist that clone's study need focus on tracking and managing clones in evolving system instead of aggressively refactoring them.

2.3 Clone Tracking

Clone genealogy is one method to help tracking and managing clones in evolving systems. To tracking clones in evolving systems, Duala-Ekoko *et al*. [1] proposed the notion of an abstract Clone Region Descriptor (CRD), which describes the clone instances within methods in such a way that it is independent of the exact text or their location in the code. Given a CRD and a code base, they develop a CloneTracker which is implemented as an Eclipse plug-in and identify the corresponding clone region through a series of automatic searches.

Göde *et al*. [9] presented an incremental clone detection algorithm, which detects clones based on the results of the previous revision's analysis. Their algorithm creates a mapping between clones of one revision to the next, basing on the information about the addition and deletion of clones. The tool of CYCLONE [10] can construct genealogies from iClones's detection results.

3. Our Approach

Our approach for tracking clone code changes consists of the following four steps:

(1) collected the clone detection result
(2) mapping the clone groups and the fragments
(3) automatic identification of change patterns
(4) visual the result in a graph

At first we use the auto detection tools to detect the clones and get the information about clones and clone groups of a software system across multiple versions. We use an auto detection tool FCD(Function Clone Detector) to detect the target system. Then we

map a certain clone from a version to the corresponding clone in the next version. Basing on the map information, we will find the change patterns which we have defined. The information can show with graphs or reports. Figure 1 shows the general roadmap of our study.

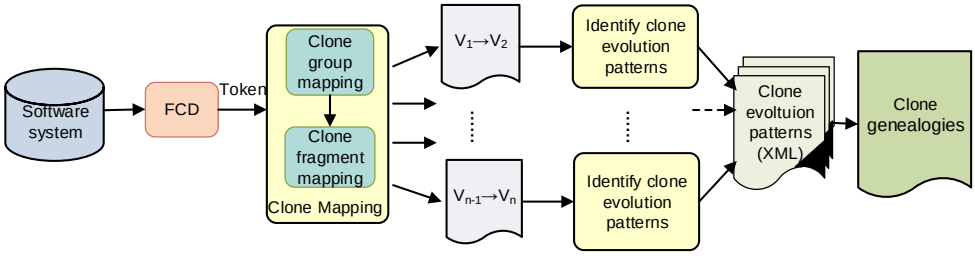

Figure 1. General Roadmap of our Approach

3.1 Clone Detection

We use the FCD tool to find the code clones and classify the clone classes. FCD use the token-based approach implemented on the optimized suffix array, it can efficiently detect type-1 and type-2 function clones, and with good recall and precision. To have a good show and map these clones, we save these results to XML files. Figure 2 shows the clone information of FCD detected.

```
<?xml version="1.0" encoding="UTF-8" ?>
- <clones>
  - <clone>
      <version>bluefish-2.2.4</version>
      <cloneID>1</cloneID>
      <cloneFrag>1</cloneFrag>
      <location>src/preferences.c</location>
      <beginline>1150</beginline>
      <endline>1167</endline>
      <functionName>filters_query_tooltip_lcb</functionName>
    </clone>
```

Figure 2. Clone Information of a Version in a XML

3.2 Clone Mapping

After clone detecting, we get clones information across multiple versions. We need map of the clone to two neighboring versions. The clone mapping results will influence the identification of clone changes patterns, because if some maps not be found, the clone chains would miss. To find effective mapping, we present a token-based clone mapping method, which map clones at two steps: (1) clone groups mapping, (2) clone fragments mapping. The reason is that we need know the clone fragments relationship to research clone changes and to greatly reduced the number of matching we first map clone groups, then map each clone fragment in these clone groups.

(1) Clone Groups Mapping: To trace a clone group across versions, we define a TokenSimilarity function, which measures how similar the clone groups are. We get two tokens T1 and T2 which token of CG1 and CG2 respectively. Then we use the longest common subsequence (LCS) algorithm to calculate the common ordered tokens between T1 and T2. Here T1 and T2 come from the auto detector FCD. Equation (1) describes the TokenSimilarity function, Where LCS(T1,T2) denotes the length of longest common subsequence of T1 and T2. Len(T) denotes the length of the token T. The value of TokenSimilarity is between 0 and 1, which reveal how similar two clone groups are in terms of lexicon.

$$TokenSimilarity(T_1, T_2) = \left\{\frac{LCS(T_1,T_2)}{len(T_1)} + \frac{LCS(T_1,T_2)}{len(T_2)}\right\}/2 \quad (1)$$

(2) Clone Fragments Mapping: The clone group mapping build the relationship of two versions' clone propagate, but to precisely track the clone changes, it is essential to research the clone fragments granularity. But how to get the clone fragments mapping on two clone group mapping? We use characteristics metric priority method to quantify the clone fragments mapping. These characteristics are aggregated from the following:

- FilenName: Name of the file containing the clone fragment
- functionName: Name of the function containing the clone fragment
- startLine: Start line number of the clone fragment in the contained file
- endLine: End line number of the clone fragment in the contained file
- size: Lines of clone fragment code

These characteristics come from the clone detector FCD. Why use these characteristics is because that the location, the file name and the function name are all maybe change when a clone code modify in next version. And a clone fragment also may be moved to different file, to this situation, the fileName be changed and the contained function may be renamed. We can know the location information from the starLine and the endLine, and know the modify condition of clone fragments function from the functionName and FileName characteristics.

To all possible matching clone fragment pairs <CF1,CF2>, their clone group map each other, so their token is similar. We depend on the different characteristics order the priority, the order rules organized from highest to lowest priority list on is Figure 3:

(1) If $CF_1.fileName = CF_2.fileName$ and $CF_1.functionName = CF_2.functionName$, match $<CF_1,CF_2>$.

(2) If $CF_1.fileName = CF_2.fileName$ and $isMatched(CF_1.functionName,CF_2.functionName)$ = true and $SizeSimilarity(CF_1,CF_2) > 0.3$, match $<CF_1,CF_2>$.

(3) If $CF_1.fileName \neq CF_2.fileName$ and $CF_1.functionName = CF_2.functionName$ and $SizeSimilarity(CF_1,CF_2) > 0.3$, match $<CF_1,CF_2>$.

(4) If $CF_1.fileName \neq CF_2.fileName$ and $isMatched(CF_1.functionName,CF_2.functionName)$ = true and $SizeSimilarity(CF_1,CF_2) > 0.3$, match $<CF_1,CF_2>$.

Figure 3. Clone Fragments Mapping Algorithm

3.3 Identify Five Change Patterns

We define and identify five change patterns based on the kim' research. The change patterns show in Figure 4.

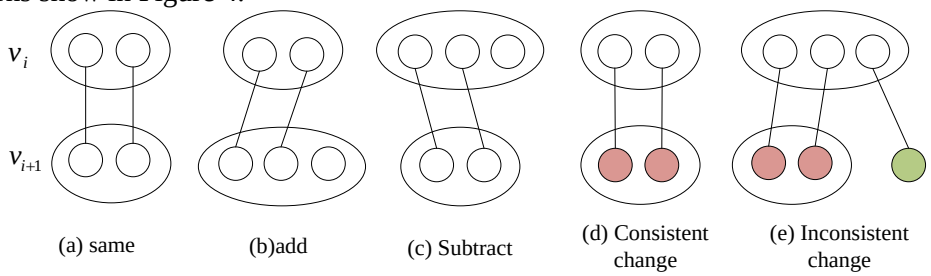

Figure 4. Five Clone Patterns

These patterns' description as Table 2 describes:

Table 2. The Five Clone Change Patterns Details

patterns	describe	Figure 4 shown
Same	all clone fragments in NG did not change from OG	a
Add	at least one clone fragment in NG is newly added	b
Subtract	at least one clone fragment in OG has changed or removed, thus it does not appear in NG	c
Consistent Change	all clone fragments in OG have changed consistently, thus they belong to NG together due to maintaining similarity	d
Inconsistent Change	at least one clone fragment in OG changed inconsistently, thus it does not belong to NG anymore. For example, one clone fragment in OG remains unchanged while others changed consistently	e

These five patterns described the every change of a clone group. The change patterns maybe have more than one happen to a clone. Thus some patterns possible have overlap. The relationship of these patterns shows in Figure 5.

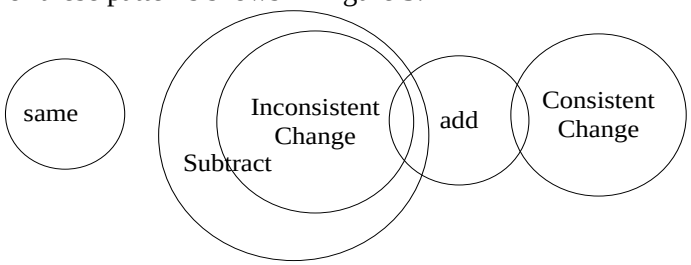

Figure 5. The Relationship of the Five Change Patterns

In this paper, we present a multi-round method to efficiently identified five evolution patterns. There are two rounds, the divided method based on the relationship of these patterns. Some patterns are exclusive each other, such as same pattern and consistent change pattern. Add pattern and subtract pattern are exclusive with same pattern. Inconsistent change pattern is exclusive with same and consistent change. So we have two rounds to identify all these five change patterns. The rounds are show in Figure 6:

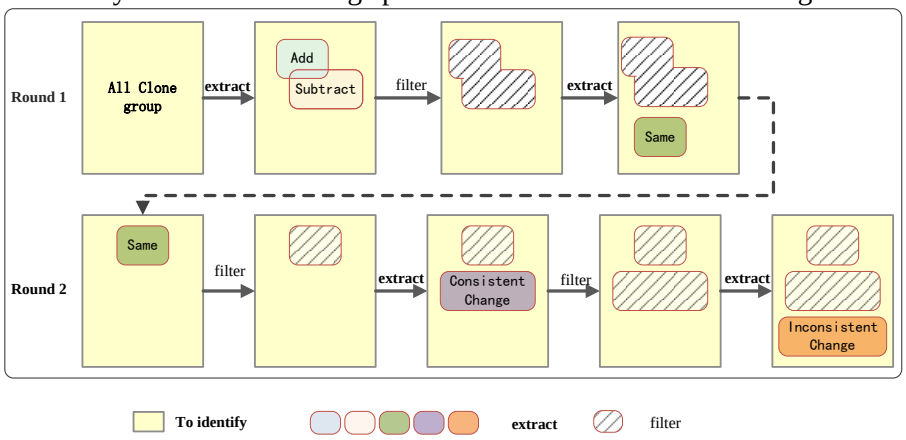

Figure 6. The Identification of the Five Change Patterns

In the first round, the add, subtract and same patterns will be marked. It has three steps, at first we extract add and subtract patterns, the method is that if some clone fragment in NG has new clone fragment appears, the clone group pattern is add. And if some clone fragment in NG is disappeared, the subtract pattern will be marked. At second, we filter

add and subtract patterns. The reason is that the same pattern is exclusive with add and subtract, so we will extract same in the remaining clone groups after filtering add and subtract. At third, we will extract same if every clone fragment in NG is same as in OG.

In the second round, the consistent change and inconsistent change will be marked. At begin, we need the all no marked clone besides same, the reason is consistent change and inconsistent change is exclusive with same. So we first filter same pattern, then marked consistent change to the clone groups in which all clone fragments have same change in OG and NG. In similar, if some clone fragment in OG has changed but it no same change in NG, it will be marked inconsistent change pattern.

3.4 Extraction and Visualization the Clone Genealogies

A Clone Lineage is a directed acyclic graph that describes the evolution history of a clone group. **A Clone Genealogy** is a set of clone lineages that have originated from the same clone group. We have built the clone mapping and identify the change patterns. Tracking these results, we will get all the clone lineages and clone genealogies. Figure 7 shows an example clone genealogy comprises two clone lineages.

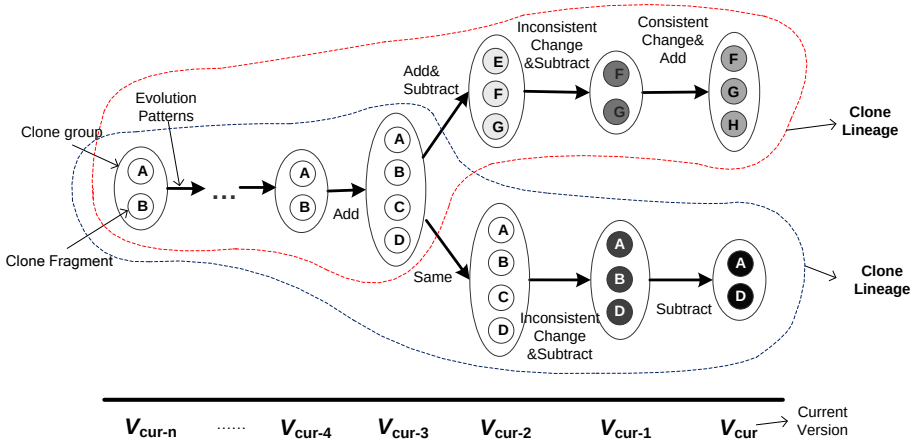

Figure 7. An Example Clone Genealogy

To get the clone lineages, we start with current version as Vcur, and tracking the adjacent version ahead. If the clone group's ID is match, then merge the clone relationship. After all the clone mapping have matched and merged, we will get the clone lineages. After merging all the clone lineages, we will get all the clone genealogies.

After finding the clone genealogies in software systems, using the auto visual tools will help tracking the clone changes. We develop a prototype tool named cGenShow. It can track the clone changes of different version and visual the change patterns. To track the clone changes, the cGenShow need the mapping information and change patterns information. To help visual these information, all these information are stored in XML files. Using the tool, we can track the adjacent version clone relationship based on the clone mapping. If the clone changes are different, the color show is variation.

4. Experiments and Results

4.1 Subject System and Parameter

To track the clone changes on the results of clone genealogies, we choice an open source HTML editor software Bluefish as subject. The system has 18 stable releases continuously updated from 2001-01 to 2014-05. The sizes of these versions range from approximately 126027 to 71232 lines of code (LOC). We set the minimum token length

to 30 for FCD, because very short code fragments have not enough practical significance. And the same value was also used in other studies previously.

4.2 Clone Evolution Patterns

We statistic the percentage of each clone group evolution pattern occur during the software evolution, as shown in Table 3. To the bluefish software, most of clone did not change during the version upgrade, and both consistent change and inconsistent change are less frequent. As for those changed clones, consistent change has a higher percentage than inconsistent change. This explains that most clones can be maintained well over the course of evolution.

Table 3. The Percentage of each Clone Group Evolution Pattern in Bluefish Software

versions (bluefish)		sum	same		add		subtract		consistent change		Inconsistent Change	
source	object	no.	no.	%	no.	%	no.	%	no.	%	no.	%
2.2.6	2.2.5	47	47	100.00	0	0.00	0	0.00	0	0.00	0	0
2.2.5	2.2.4	45	42	93.33	1	2.22	1	2.22	1	2.22	0	0
2.2.4	2.2.3	46	43	93.48	2	4.35	1	2.17	0	0.00	0	0
2.2.3	2.2.1	43	40	93.02	1	2.33	0	0.00	2	4.65	0	0
2.2.1	2.0.3	32	20	62.50	5	15.62	7	21.88	0	0.00	0	0
2.0.3	2.0.2	29	20	68.97	4	13.79	4	13.79	1	3.45	0	0
2.0.2	2.0.0	23	20	86.95	2	8.70	1	4.35	0	0.00	0	0
2.0.0	1.0.7	14	4	28.57	5	35.71	5	35.71	0	0.00	0	0
1.0.7	1.0.6	17	17	100.00	0	0.00	0	0.00	0	0.00	0	0
1.0.6	1.0.5	17	16	94.00	0	0.00	0	0.00	0	0.00	1	6
1.0.5	1.0	18	16	88.89	1	5.56	1	5.56	0	0.00	0	0
Total		331	285	86.40	21	6.34	20	6.04	4	1.21	1	0.003

4.3 Clone Changes Frequently

Figure 8 shows that all the genealogies over their age. The age of a clone genealogy is the number of versions that the genealogy spans. It can indicate how long the clone sustained. In bluefish, the number of clone lineages is 81 and the average lifetime of genealogies alive is 6.56.

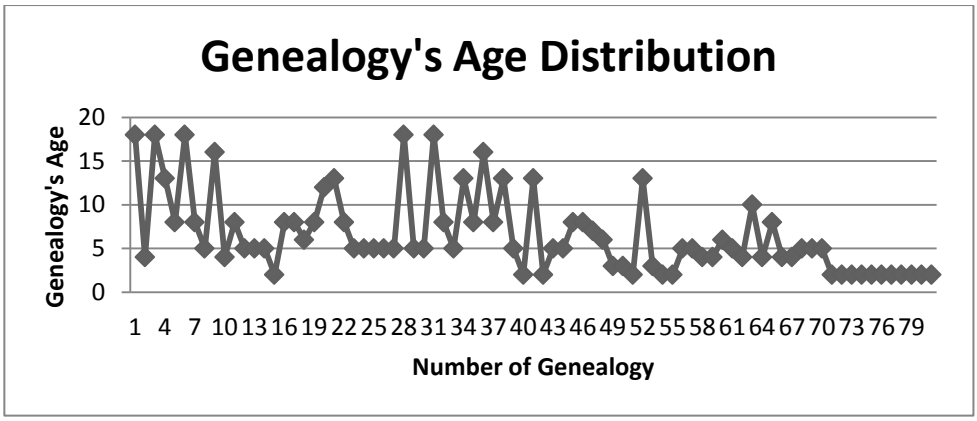

Figure 8 .Clone Changes Frequently

4.4 Clone Changes Tracking

To track the clone changes, We develop a prototype tool named cGenShow. It can show every clone changes. The example of cGenShow is shown in Figure 9. We can get the details about the clone mappings, the clone version, the clone group, the clone position and the clone code. In the visual form, we can find the clone changes through the different color.

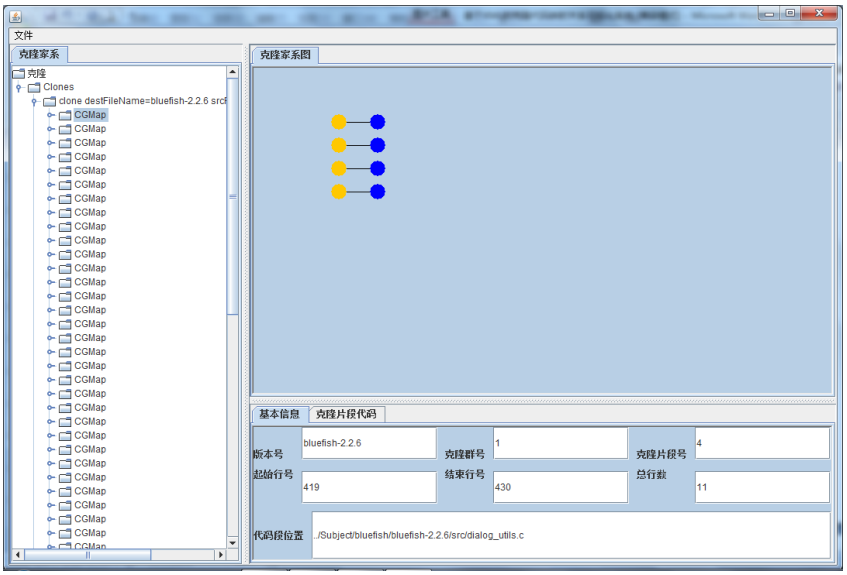

Figure 9. Clone Change Tracking

5. Conclusion

In this paper, we track the clone change by identifying clone evolution patterns and extracting clone genealogy. At first, we get the clones and clone groups' data from the clone detector FCD. Then to every two adjacent versions constructs the mapping. At last, basing on the code clone's changes in clone groups in two adjacent versions, we can identify the clone evolution patterns and get the clone genealogy.

Basing on the above method, we implement the tools of clone genealogy extractor cGen and clone genealogy show cGenShow. Using these tools we performed an experiment on a software system Bluefish. The system has 18 stable releases continuously updated from 2001-01 to 2014-05. The sizes of these versions range from approximately 126027 to 71232 lines of code. After using the above method, we get the changes

information about code clones. These information is helpful to find code clone change patterns, changes frequency and tracking the changes attribute and so on.

Our method is fit for type-1 and type-2 clone, and the programming language is only fit for C. the reason is that our clone detector is basing on the token, and the lexical analyzer is for C and the token method is effective to type-1 and type-2. However, the method is extensive to other programming language and detector types. These studies are our subsequent tasks.

Acknowledgements

This work was supported by the National Natural Science Foundation of China under Grant No.61363017 and No.61462071, the Natural Science Foundation of Inner Mongolia under Grant No.2014MS0613, and the Scientific Studies of Higher Education Institution of Inner Mongolia Autonomous Region of China under Grant No.NJZY14039.

References

[1] E Duala-Ekoko, M P Robillard, "Tracking code clones in evolving software", Proceedings of the 29th International Conference on Software Engineering. IEEE Computer Society, **(2007)**, pp. 58-167.
[2] T.Kamiya, S.Kusumoto, K. Inoue. "CCFinder: A Multi-linguistic Token-based Code Clone Detection System for Large Scale Source Code" IEEE Transactions on Software Engineering, vol. 28, no. 7, **(2002)**, pp. 654-670.
[3] J R Cordy, C K Roy, "The NiCad clone detector", Proceedings of the 19th International Conference on Program Comprehension". IEEE, **(2011)**, pp. 219-220.
[4] M. Uddin , C K Roy, K A Schneider, "Simcad: An extensible and faster clone detection tool for large scale software systems", Proceedings of the 21st International Conference on Program Comprehension. IEEE, **(2013)**, pp. 236-238.
[5] J. Harder and N. G¨ode, "Modeling clone evolution", Proc. IWSC, **(2009)**, pp. 17–21.
[6] M Kim, V Sazawal, D Notkin, *et al*, "An empirical study of code clone genealogies[C]"//ACM SIGSOFT Software Engineering Notes. ACM, vol. 30, no. 5, **(2005)**, pp.187-196.
[7] R K Saha, C K, Roy, K A Schneider. "gcad: A near-miss clone genealogy extractor to support clone evolution analysis[C]"//Proceedings of the 29th International Conference on Software Maintenance. IEEE, **(2013)**, pp. 488-491.
[8] N Göde, R Koschke, "Frequency and risks of changes to clones", Proceedings of the 33rd International Conference on Software Engineering. ACM, **(2011)**, 311-320.
[9] N Göde, R Koschke, "Incremental clone detection. In: European Conference on Software Maintenance and Reengineering", **(2009)**, pp. 219–228 .
[10] J. Harder and N. Göde, "Efficiently Handling Clone Data: RCF and cyclone," Proc. IWSC, (**2011**), pp. 81–82.

Authors

Chunhui Wang, She Received the BS and MS degrees in computer science from Inner Mongolia Normal University, China, in 2002 and 2008. Currently, her research interests include software analysis, multi-media and CAI.

