International Journal of Security and Its Applications
Vol. 11, No. 3 (2017), pp.81-94
http://dx.doi.org/10.14257/ijsia.2017.11.3.08

Automatic Extract Clone Genealogy to Analysis Software Evolution Process

Chen Zhuo[1], Zhang Li-ping[2*] and Hou Min[3]

Department of Computer & information Engineering
Inner Mongolia Normal University
[1]18748156429@163.com, [2]cieczlp@imnu.edu.cn,
[3]houmin920@163.com

Abstract

Clone code is duplicate or similar code fragment, these duplicate codes from the copy and paste operation, such code are believed to reduce the maintainability of software severely. In this paper, we developed a clone genealogy extractor-ECG, which can accurately and quickly build Type-1, Type-2 and Type-3 clone genealogy. We examined four open source software, found that about 42% of clone code was not changed in the evolution process, and about 3.48% of clone code had inconsistent change, such clones may introduce potential bugs that need to be focused on. Experiments results show validity and high efficiency of clone genealogy extractor, which provides reference and data support for code clone quality assessment and management.

Keywords*: code clone; clone mapping; evolution pattern; clone genealogy; evolution analysis*

1. Introduction

Developers often copy and paste code in the process of software system development[1], The reuse code is called Clone Code [1]. Existing studies have shown that software system contains a lot of clone code. About 9%~17% of clone code exists in software systems[2], the proportion of clone code in industrial software is higher, and the clone code is also regarded as a potential threat to software maintenance and development. Early studies suggest that clone code is harmful, and thus refactoring the clone code. Since then, the researchers have found that cloning should be analyzed and evaluated, rather than blindly reconstructed. In the process of analyzing the clone code, clone evolution and clone genealogy are two core questions. Therefore, understanding the clone code evolution is essential to clone code management system.

At present, there are a lot of researches on clone evolution, and mainly focused on the Type-1, Type-2 clone code, and the evolution research of the Type-3 clone code is relatively few. So it is very important to accurately extract the Type-1, Type-2 and Type-3 types of clone genealogy and further understand evolutionary patterns.

In this paper, we present an approach to extract the clone genealogy automatically. First, we obtain the detection information by clone detection tools. Then to mapping clone relationship according to clone features and word frequency vector. And identify the clone evolution patterns according to mapping results. At last, we can automatically extract the clone genealogy. The approach can provide basic data for later application research.

ISSN: 1738-9976 IJSIA

2. Related Work

2.1 Definition and Classification of Clones

The definition of code clone is widely adopted as similar syntax and semantic features of code segment at present [3]. A clone pair is the two fragments that are similar to each other in the same version of system. Two or more similar code segments form a clone group. Clone group mapping reflects the change process from a previous version to the current version. In current research, there are two main classification approaches [4]: similarity degree and granularity of code segment. According to the text similarity of source code, the clone is divided into Type-1 to Type-4 clone. Type-1 to Type-3 clone reflects the degree of similarity in grammar, and Type-4 reflects the semantic similarity. The size of the code segment is divided into files, blocks, functions, classes and statements.

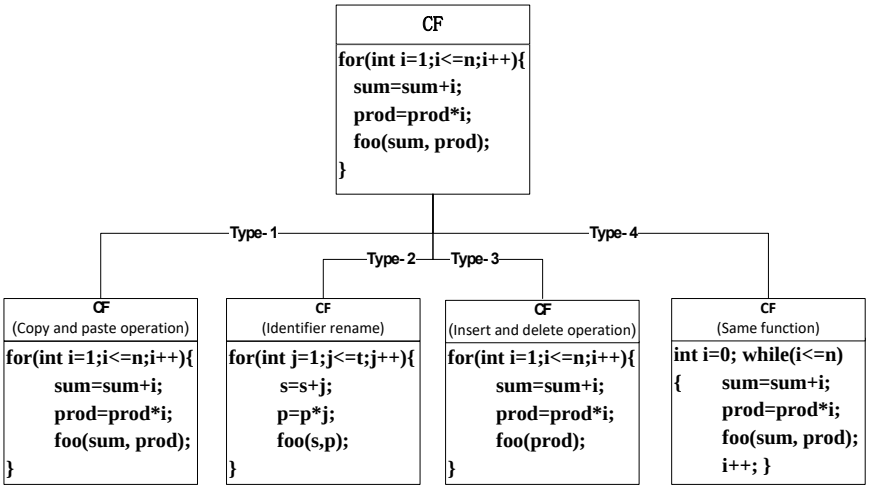

Figure 1. Clone Type Definition and Classification

2.2. Clone Tracking

Cloning mapping needed to be established before extract clone genealogy, the accuracy of clone mapping directly affects the whole study results. Therefore, it is very important for clone evolution study.

Hotta *et al*. [5] proposed an approach to track the clone in evolutionary system. According to the clone region descriptors (CRD), they described the clone mapping relationship from the clone text content and the file location. Although the approach could map clone that had changed positions, the false positive rate is higher.

Bakota *et al*. [6] proposed a mapping approach based on abstract syntax tree, this approach mapped clone according to the file name, location, and clone distances. Although this approach can map clones in multiple versions, but a large number of similar features increase their time consumption.

Thummalapenta S *et al*. [7] proposed an approach based on the modified log to tracking clone relationship. To obtain system modification log from the CVS code library, the first version as the origin, to calculate the changes in the subsequent version, so as to get the mapping relationship. However, due to the origin of the version of the standard, so the new clone cannot be studied in the subsequent version.

Saha *et al*. [8] conduct the function mapping between versions according to the name and file path, then to map clone from the detection results. Although this approach can improve the running time, it is easy to be affected by the change in the position of the clone.

2.3. Clone Evolution and Clone Genealogy

The single version of the software does not contain the evolutionary relationship of clone code. We need a better understanding of the variation of the clone code in multiple versions.

Kim *et al*. [9] proposed the most representative clone genealogy framework. The clone genealogy can be used to describe the variation of clone group in multiple versions. Kim *et al*. develop a clone genealogy extractor CGE to get a software systems clone genealogy. 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*. [10] developed a clone genealogy extractor—gCad. They use the TXL extraction function, then to mapping clone between adjacent versions according to the function of the signature function, class name and file name attribute. If the function is renamed or moved to a different file or directory, it is based on the similarity of the name and content of the function to find the origin. After all functions are mapped, the problem scale is reduced to the function mapping from the version mapping, So the calculation speed is very fast。But the lack of the study is that if the clone fragment is moved to other functions, it cannot be accurately mapped.

3. Our Approach

Our approach for extract clone genealogy consists of the following four stages:

1）obtain the clone detection result

2）clone mapping（clone group and clone fragment）

3）classification and identification of clone evolution patterns

4）extraction clone genealogy

Figure 2 shows the general roadmap of our study.

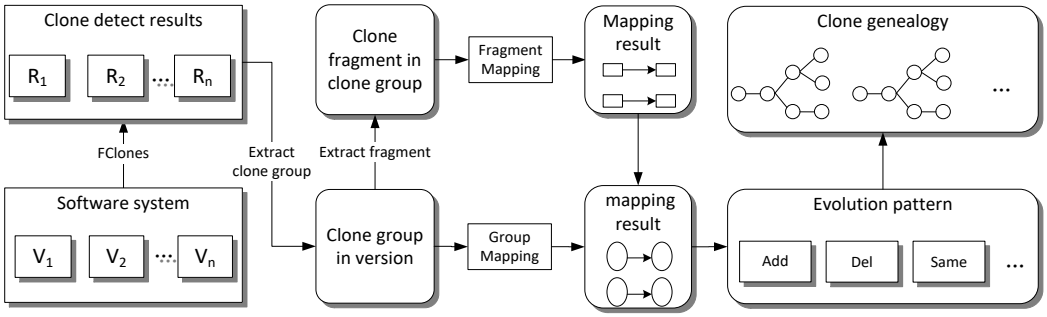

Figure 2. General Roadmap of our Approach

3.1. Clone Detection

We use the FClones [11] tool to find the code clones and classify the clone group. FClones use the token-based approach implemented on the edit distance, it can accurately detect the Type-1, Type-2 and Type-3 clone code. This tool uses multiple versions of the software as input, to get the file name, the function name, start line, end line and other the relevant information of clone fragment. These data are stored in the XML file, in order to facilitate the extraction and treatment of late study. Figure 3 shows the clone detection results.

```
<CloneGroup id="1" clones="2">

<source file="D:\___Clones\OpenSource\Software\000bluefish\bluefish-1.0.6\src\bf_lib.c"
        startLine="1433" endLine="1441" tokenID="178">
</source>
<sourcecode>
<![CDATA[
    if(*chars > 0) (*lines)++;
}
GSList *gslist_from_glist(GList *src) {
    GSList *target=NULL;
    GList *tmplist = g_list_first(src);
    while (tmplist) {
        target = g_slist_append(target, tmplist->data);
        tmplist = g_list_next(tmplist);
    }
    return target;
}
]]>
</sourcecode>
```

Figure 3. Clone Detection Results

3.2. Clone Mapping

we get clones information across multiple versions after obtaining the clone detection results. The number of clone fragment is far more than the clone group in the software version. So it can reduce the number of the map to a large extent by clone group maps. So it is first to the clone group mapping, and then to map the inner clone fragment, this can greatly reduce the number of mapping.

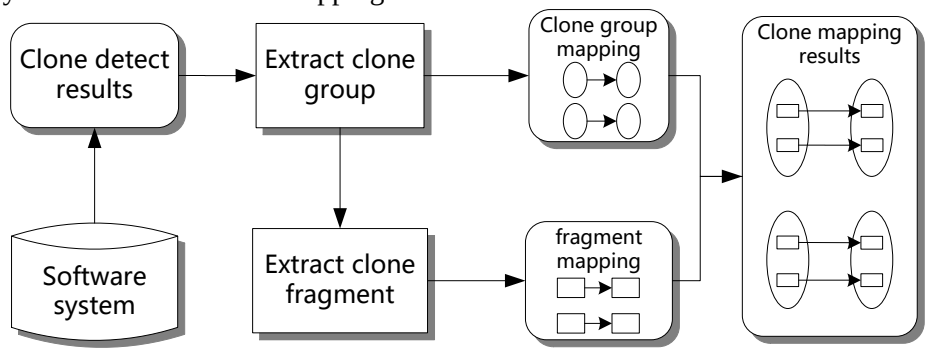

Figure 4. Clone Mapping Process

1）Clone group mapping stage

In this part, we use the word frequency vector and clone feature to mapping clone group. First we calculate the similarity of the clone group by word frequency vector, it will serve as candidate clones if meets threshold conditions. Then to match the clone group feature, features include the file name, start line, end line, and the number of code lines. Finally, the mapping relation of the clone group is determined by matching those clone features.

The similarity cosine algorithm is used to calculate the similarity degree between the word fragment vectors of the clone group. Suppose the word fragment vectors of the two clones are respectively ω_i and ω_j. *CosSimilarity* (ω_i, ω_j) algorithm is used to calculate their similarity.

$$CosSimilarity(\omega_i, \omega_j) = \frac{\sum_{i,j=1}^{n} \omega_i * \omega_j}{\sqrt{\sum_{i=1}^{n} \omega_i^2 \sum_{j=1}^{n} \omega_j^2}} \quad (1)$$

$\omega_i * \omega_j$ refers to the inner product of word fragment vector. The range of the value of *CosSimilarity* is [0, 1]. In this paper, we set up a threshold *t*, all of the *CosSimilarity*>=t clones were used as clone groups with mapping relationships.

2）Clone fragment mapping stage

First of all, the similarity calculations of the clone fragment by word frequency vector. To determine the mapping relationship between clone fragment by matching code line distance and clone fragment feature. In this paper, we define the *LocDistance* used to measure the code distance of clone fragment. In general, although the Type-3 clone code will be modified, but the changes will not be too large, Even if the code is to add or delete operations, the number of line changes will not be greater than the half of the total number of code lines. Therefore, it does not consider a mapping relationship if the number of code lines is a big difference.

$$LocDistance(CF_i, CF_j) = \frac{\min(Loc.CF_i, Loc.CF_j)}{\max(Loc.CF_i, Loc.CF_j)} \quad (2)$$

In addition, we define the *CffSimilarity* to match the clone feature. These features include: 1）Name of the file containing the clone fragment; 2）Name of the function containing the clone fragment;3）The start line of the cloned fragment；4）The end line of the cloned fragment; 5）The code line distance of clone fragment. The file name and the line number can be extracting from clone detection results, the function name can be extract from the cloned fragment in source code. After calculating the similarity of the clone fragment, we will complete the clone fragment mapping by matching the positional relations and features of the clone fragments.

3.3. Clone Evolution Pattern Identification

First, we classify the individual clone fragment, and then classify and identify clone evolution patterns base on code fragment. Due to the clone group contain a number of clone fragments, so it can identify clone evolution patterns in the stages, which can greatly reduce the recognition times and improve the recognition efficiency.

1）Clone Fragment Evolution Patterns

According to the changes of the number and the content of the cloned fragments, the changes of the clone fragments in the clone group with mapping relations are divided into four kinds, which are add, remove, unchanged and change:

- **Appear** (distinguish from quantity): Clone fragment *CF* was new created in version V_{i+1}, which does not exist in version V_i.
- **Disappear** (distinguish from quantity): Clone fragment *CF* exists in version V_i, but disappears in version V_{i+1}.
- **Unchanged** (distinguish from content): The content of clone fragment *CF* without any changes from version V_i to version V_{i+1}.
- **Change** (distinguish from content): The content of clone fragment *CF* changed from version V_i to version V_{i+1}.

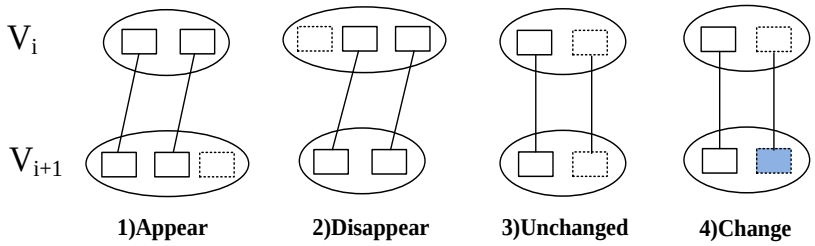

Figure 5. Clone Fragment Evolution Patterns

2）Clone Group Evolution Patterns

This paper is concerned with the changes in the content and quantity of the clone code. Because the location of the Type-3 clone code is easy to change with the code deletion and increase operation. Therefore, it is impossible to provide accurate reference for clone evolution study, so the shift pattern is not used as the research content。In addition, we find that there exist separation and merging phenomena when observe the real change of the clone code during the evolution process, thus increasing the *Separate* and *Merge* evolution pattern in this paper. In order to meet the needs of the analysis of the late evolution of clonal lineages, At the same time, according to the changes of the number and the contents of the clones, the clones were divided into eight short-term evolution patterns *Static*, *Add*, *Subtract*, *Same*, *Separate*, *Merger*, *Consistent* and *Inconsistent*.

- ***Static:*** The number and contents of the clone fragment in the clone group *CG* were not changed from version V_i to version V_{i+1}.
- ***Add:*** At least one clone fragment in clone group CG was added from version V_i to version V_{i+1}.
- ***Subtract***: At least one clone fragment in clone group CG was removed from version V_i to version V_{i+1}.
- ***Separate***: A clone group CG split into CG1 and CG2 after experiencing change from version V_i to version V_{i+1}。
- ***Merge***: Two clone groups CG_1 and CG_2 merged into CG after experiencing change from version V_i to version V_{i+1}。
- ***Same***: the number of clone fragment in clone group CG remains the same from version V_i to version V_{i+1}。
- ***Consistent***: The contents of all the clone fragments in clone group CG have changed consistently from version V_i to version V_{i+1}.
- ***Inconsistent***: At least one clone fragments in clone group CG have changed inconsistently from version V_i to version $V_{i+1,}$ lead to no longer belong to the same clone group.

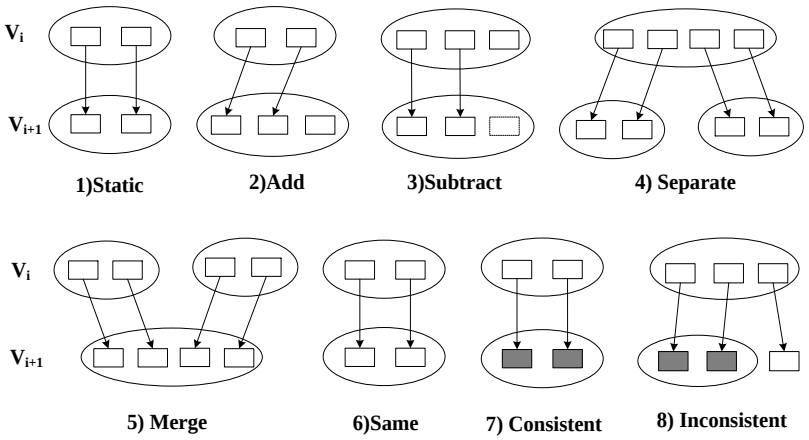

Figure 6. Clone Group Evolution Patterns

Clone evolution pattern was added according to the content and the quantity change of the clone fragment. It assumes that there is a clone mapping relationship between the clone group CG_i and CG_{i+1}, then the specific identification process:

Step1：If the number of clone fragments in the clone group CG_i and CG_{i+1} does not change, then to mark "Same" pattern. If the clone group CG_{i+1} contains new fragment，then mark "*Add*" pattern. If at least one clone fragment in clone group CG_i has removed，then mark "*Subtract*" pattern.

Step2：If the content and quantity of clone fragments does not change, then to mark "*Static*" pattern. If the content of all clone fragments have changed consistently, then mark "*Consistent*" pattern. If the content of all clone fragments have changed inconsistently, then mark "*Inconsistent*" pattern.

Step3：If two clone group in the version V_{i+1} are derived from the clone group CG_i in the version V_i, then mark "Separate" pattern. If two clone group in the version V_i correspond to the clone group CG_i in the version V_{i+1}, then mark "*Merge*" pattern.

3.4. Clone Genealogy Extraction

A Clone Lineage describes the evolution of a clone group that no separation occurs during the evolution period. As shown in Figure 7, the clone group CG_{i-1}^1 in version V_{i-1} has the mapping relationship with the clone group CG_i^1 in version V_i (CG_{i-1}^1 is the source clone group, and CG_i^1 is the target clone group). The clone group CG_i^1 in version V_{i-1} has the mapping relationship with the clone group CG_{i+1}^2 in version V_i(CG_i^1 is the source clone group, and CG_{i+1}^2 is the target clone group). To link the clone group ID of each version, so as to get a Clone Lineage $CG_{i-1}^1 \rightarrow CG_i^1 \rightarrow CG_{i+1}^2$.

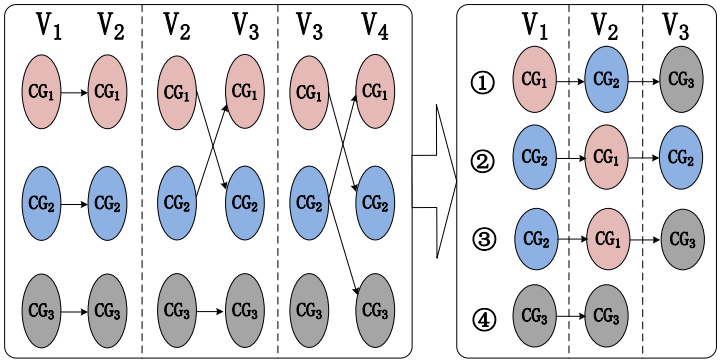

Figure 7. Clone Lineage Construction Process

Each clone lineage has the corresponding attribute, which includes the version ID and clone group ID of source clone group. We identify all the clone lineages that originated from the same clone group after getting all clone lineages, and then to construct the clone genealogy. **Clone genealogy** describes a set of clone lineage that originated from the same clone group. Figure 8 shows the process of building clone genealogy.

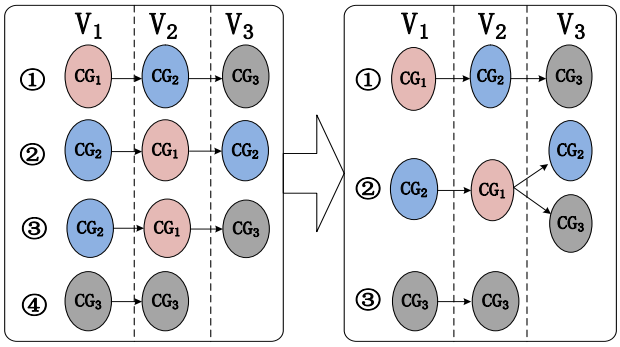

Figure 8. Clone Genealogy Construction Process

The clone genealogy can reflect the evolutionary process of the clone group during the software life period, a clone genealogy reflects how the clone code is created, propagated, and changed. Figure 9 describes the clone genealogy with two clone lineages. There are two clone lineages in the graph, the V_i indicates the version number, the arrow indicates the mapping relationship. The shape change of the clone fragment indicates that the content of the fragment is changed.

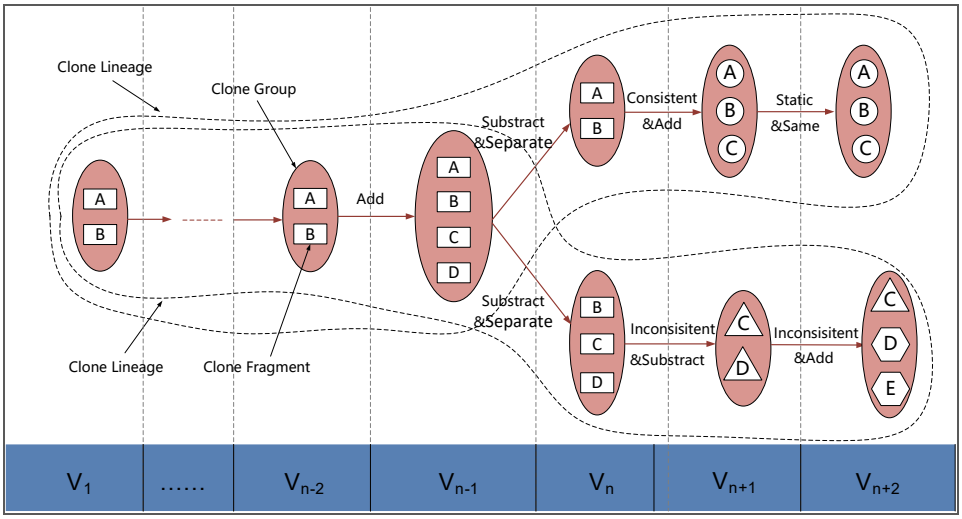

Figure 9. Clone Genealogy Sample Graph

We store clone genealogy in the XML file, clone genealogy information includes：The version number of the file, mapping relationship between clone groups, clone evolution pattern and clone fragments information. Figure 11 shows the storage way of clone genealogy.

```
<CloneGenealogy ID="1">

<SystemInfo File="Bluefish" Tool="ECG" System="win64" Granularity="CloneGroup" Threshold="0.85"/>
<CloneCountInfo CodeFragmentCount="2280" CloneGroupCount="594"/>
<CloneEvolution Id="1">
    <VersionInfo SrcVersion="Bluefish-1.0.5" DestVersion="Bluefish-1.0.6"/>
    <GroupEvoInfo SrcCGId="1" DestCGId="1" EvolutionPattern="Static+Same">
        <FragmentEvoInfo SrcCFId="1" DestCFId="1"></FragmentEvoInfo>
        <FragmentEvoInfo SrcCFId="2" DestCFId="2"></FragmentEvoInfo>
        <FragmentEvoInfo SrcCFId="3" DestCFId="3"></FragmentEvoInfo>
        <FragmentEvoInfo SrcCFId="4" DestCFId="4"></FragmentEvoInfo>
    <GroupEvoInfo>
</CloneEvolution>

<CloneEvolution Id="2">
    <VersionInfo SrcVersion="Bluefish-1.0.6" DestVersion="Bluefish-1.0.7"/>
    <GroupEvoInfo SrcCGId="1" DestCGId="1" CGPattern="Subtract">
        <FragmentEvoInfo SrcCFId="1" DestCFId="1"></FragmentEvoInfo>
        <FragmentEvoInfo SrcCFId="2" DestCFId="2"></FragmentEvoInfo>
        <FragmentEvoInfo SrcCFId="3" DestCFId="3"></FragmentEvoInfo>
        <FragmentEvoInfo SrcCFId="4" DestCFId="N/A"></FragmentEvoInfo>
    <GroupEvoInfo>
</CloneEvolution>
```

Figure 11. The Storage form of Clone Genealogy

As shown in the figure 11, the first label (the first line) is the clone genealogy ID. The second label shows the system and the environment running information (file name, tool name, clone granularity and threshold). The third label shows the total number of clone group and the total number of clone fragment. Next is the clone group evolution of the adjacent versions of the, including the software version number, clone group Id, clone evolution pattern and the corresponding relationship between clone fragments.

4. Study Results

4.1. Subject Systems

we implemented a tool for automatic extract clone genealogy ECG according to the above method. We selected four C language open source software, a total of 64 release versions as the experimental data. This experimental target software from different areas, different sizes and has continued to update, in order to ensure the accuracy and reliability of the experiment. Table 1 shows the information of the software for the experiment.

Table1. Subject System

Subject System	Bluefish	FFmpeg	Emacs	Lighttpd
Start time	2001-01	2013-09	2005-02	2007-04
End time	2014-05	2014-07	2013-03	2014-03
No. of version	18	14	11	21
Line of code	26027~71232	319002~739058	318258~353694	51691~54520
Average size	250M	83M	77M	72M
Area	Html Editor	Framework	Text Editor	Web Server

We obtain the clone code of software system through clone detect tool—FClones, This tool detection method based on Token edit distance, It can effectively detect the Type-2, Type-1 and Type-3 clone code in the open source software, And the detection results are stored in the XML file, which is convenient for extract and use the data.

4.2. Experimental Parameters

The experiment was run in the Windows environment, Specific environment configuration: Intel(R) Core(TM) i5-3210M CPU @2.50GHz，8G memory. The experiment takes the version number as the time line, then tracks the mapping relationship and clone changes. We have done a lot of comparative experiments and verify the precision of the threshold in [0.6, 1.0] and 0.05-intervals. Figure 12 shows the validation results under different thresholds.

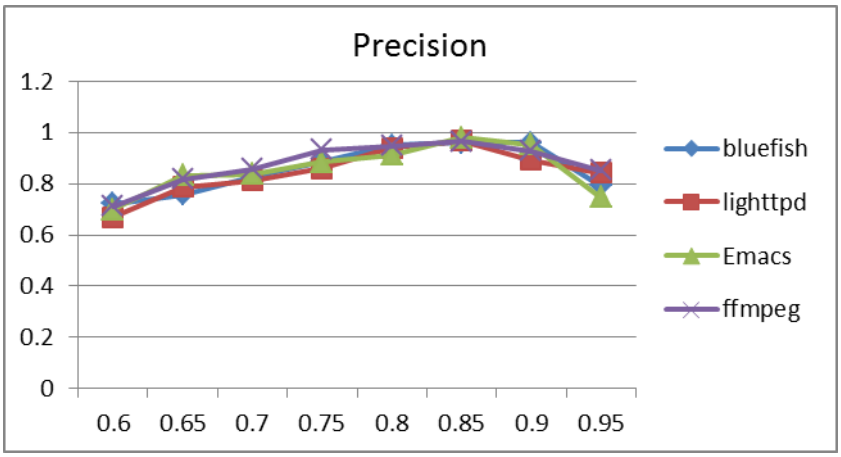

Figure 12. Validity Results Under Different Threshold t

After manual validation, we found that the similarity threshold value is set to 0.85, to achieve the best mapping effect, so the map ping similarity range is set to [0.85,1.0]. Threshold setting is determined by the mapping results of the specific software, so when the software system is changed, the similarity threshold needs to be re-verified.

4.3. Clone Group Evolution Patterns

In this paper, we study the Type-1, Type-2 and Type-3 function clone, and conducted experiments on 64 versions in 4 open source software. We count the number of each clone group evolution pattern, and calculate the proportion of each clone evolution pattern during the software evolution. Table 2 shows the statistical results of clone group evolution pattern.

Table 2. Proportion of Clone Group Evolution Pattern

Subject System	Static	Same	Add	Subtract	Separate	Merge	Consistent	Inconsistent
Bluefish	38.60%	45.70%	3.20%	2.20%	1.90%	1.10%	3.10%	4.20%
FFmpeg	41.90%	48.20%	1.13%	0.87%	0.82%	0.49%	2.71%	3.88%
Emacs	41.40%	47.90%	0.63%	1.47%	0.79%	0.42%	3.95%	3.44%
Lighttpd	46%	48.20%	0.89%	0.57%	0.72%	0.34%	0.89%	2.39%
Avg	41.98%	47.5%	1.46%	1.28%	1.06%	0.59%	2.66%	3.48%

We can see that the proportion of static evolution mode is about 41.98%, it indicates that most of the clones did not change during the update process. In addition, the proportion of consistent and inconsistent changes is about 2.66% and 3.48%, both types of evolution patterns occur in less in all software versions, and the proportion was slightly higher, it indicates that some clones are not consistently maintained in the software evolution process. Therefore, we should pay more attention to those clones who may introduce Bugs potential changes due to inconsistent changes.

4.4. Clone Genealogy Information

In addition, we also statistics the relevant information about clone genealogy, information including: the number of clones, the number of clone genealogy, the number of clone group, the number of clone fragments, life period, run time and so on. Table 3 shows the information about clone genealogy.

Table 3. Information about clone genealogy

Subject System	Number				Life Period （No. of Version）	Run Time(ms)
	Clone Group	Clone Fragment	Clone Lineage	Clone Genealogy		
Bluefish	594	2280	508	225	2~18	802
FFmpeg	5737	12212	4516	1926	2~14	9202
Emacs	1277	2724	823	391	3~11	1182
Lighttpd	1795	3182	916	474	4~21	2199

As can be seen from table 3, there will be new clone group in the software system， For example, there are 594 clone groups in the software Bluefish, and there are 508 clone lineages, it is obvious that 86 clone groups have been added during the software evolution. These new clones can reflect the software maintenance activities to some extent, the more the number, the greater the software maintenance modification activities. For example, new functions are added or have been modified to a greater extent (e.g. refactoring) in the software update process. Because if just a few modifications, clone code changes will not be too great. Larger maintenance activities will affect the clone code in the software, resulting in instability in the software evolution.

5. Conclusion

In this paper, we use word frequency vector, code line distance and clone features to map clone between multiple versions. Then classify and identify clone evolution patterns from the angles of clone group and clone fragments, so as to construct the clone genealogy. We implemented an automatic extraction tool for clone genealogy—ECG. The tool can automatically identify clone evolution patterns and extract Type-1, Type-2 and Type-3 clone genealogy. The experimental results show that the tool ECG can efficiently extract the clone genealogy. At the same time, our research results can provide data and reference for clone evolution analysis and clone management.

The clone mapping threshold is influenced by software and human factors, so it may affect the precision of clone genealogy extraction. In addition, there is no visualization of the clone genealogy after storage. These problems will be solved gradually in the future research work. The clone information is displayed to the user through the graphical method, and it can provide valuable reference for clone management and maintenance.

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 NJZY16045.

References

[1] H. A. Nguyen, T. T. Nguyen, N. H. Pham, J. Alkofahi. T. N. Nguyen, “Clone management for evolving software”. IEEE Transactions on Software Engineering, vol. 38, no. 5, **(2012)**, pp. 1008-1026.
[2] M. F. Zibran and C. K. Roy, “The road to software clone management: a survey”, Technical Report 2012-03, Department of Computer Science, University of Saskatchewan, Canada, **(2012)**
[3] Rattan, Dhavleesh, R. Bhatia, and M. Singh. "Software clone detection: A systematic review." Information & Software Technology, vol. 55, no. 7, **(2013)**, pp. 1165-1199.
[4] Q. Q. SHI, F. J. Meng, L.P. Zhang, D.S. Liu, “Survey of research on code clone technique”, Application Research of Computers, vol. 30, no. 6, **(2013)**, pp. 1617-1623
[5] Hotta, Keisuke, Y. Higo, and S. Kusumoto. "Clone Tracking based on Similarity of CRD." Technical Report of Ieice Ss 113**(2013)**.
[6] T. Bakota, R.Ferenc, & T. Gyimothy, (2007). “Clone Smells in Software Evolution”, Proceedings of the 23rd International Conference on Software Maintenance (ICSM), IEEE, **(2007)**, pp.24 - 33.
[7] S. Thummalapenta, L. Cerulo, L. Aversano, M.D. Penta, “An empirical study on the maintenance of source code clones”, Empirical Software Engineering, vol. 15, no. 1,**(2010)**, pp.1-34.
[8] R. K.Saha, M. Asaduzzaman, M. F. Zibran, & C. K. Roy. “Evaluating Code Clone Genealogies at Release Level: An Empirical Study”, Proceedings of the 10th IEEE Working Conference on Source Code Analysis and Manipulation (SCAM), IEEE, **(2010)**, pp.87-96.
[9] M. Kim, V. Sazawal, D. Notkin and G. C. Murphy, “An empirical study of code clone genealogies”, ACM SIGSOFT Software Engineering Notes. ACM, vol. 30, no.5,**(2005)**, pp.187-196.
[10] R. K. Saha, C. K. Roy and K. A. Schneider, “gCad: A Near-Miss Clone Genealogy Extractor to Support Clone Evolution Analysis”. Proceedings of the 29th International Conference on Software Maintenance (ICSM), IEEE, **(2013)**, pp.488-491.
[11] J. J. Zhang, C. H. Wang, D. S. Liu, L. P. Zhang, “Code clone detection based on levenshtein distance of token”, Computer Application, vol. 35, no. 12, **(2015)**, pp. 3536-3543.

Authors

Chen Zhuo, he was born in 1989, and he is a graduate student at Inner Mongolia Normal University. His research interests include code clone analysis and clone evolution.

Zhang Li-ping, she was born in 1974. She receives her M.S. degree from Inner Mongolia Normal University in 2005. She is an professor at Inner Mongolia Normal University. Her research interests include software engineering and software analysis.

Hou Min, she was born in 1973. She receives her M.S. degree from Inner Mongolia Normal University in 2009. She is a lecture at Inner Mongolia Normal University. Her research interests include software engineering and code analysis.