Visualizing the Evolution of Code Clones

Ripon K. Saha   Chanchal K. Roy   Kevin A. Schneider
Department of Computer Science, University of Saskatchewan
Saskatoon, SK, Canada S7N 5C9
{ripon.saha, chanchal.roy, kevin.schneider}@usask.ca

ABSTRACT

The knowledge of code clone evolution throughout the history of a software system is essential in comprehending and managing its clones properly and cost-effectively. However, investigating and observing facts in a huge set of text-based data provided by a clone genealogy extractor could be challenging without the support of a visualization tool. In this position paper, we present an idea of visualizing code clone evolution by exploiting the advantages of existing clone visualization techniques that would be both scalable and useful.

Categories and Subject Descriptors

D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement—*Restructuring, reverse engineering, and reengineering*

General Terms

Measurement, Experimentation

Keywords

Clone evolution, visualization, scatter plot

1. MOTIVATION

Despite a decade of active research, it is still not clear whether code clones are harmful in the maintenance and evolution of software systems. There are a good number of recent studies which contradict themselves on this issue and call for further research. What the community agrees is that we need to extensively study the evolution of clones should we want to study the impacts of clones in software maintenance. It is also commonly agreed that whether clones are useful or harmful, we need an efficient and cost-effective way to mange the clones. This again calls for studying the evolution of clones because without studying the evolutionary behaviour of clones, it is indeed not possible to have a good clone management system [2]. For example, if there is a bug in a clone fragment, the programmer may want to examine whether all the fragments similar to it have been changed consistently in the past, or for example, before refactoring a particular clone class, the programmer may want to see how that clone class has been changed from its birth to predict its future behaviour. Programmers also may want to examine the inconsistently changed clone classes of a particular file or directory to check whether those *inconsistent changes* have been made intentionally or accidentally. Considering the importance researchers also study the evolution of clones to draw hypotheses; to develop new tools and techniques for managing clones properly. However, an enormous amount of data (mostly textual) reported by a clone genealogy extractor makes those tasks challenging both for programmers and researchers. We believe that effective visualization of code clone evolution in terms of change patterns can help tackle these issues promptly and efficiently.

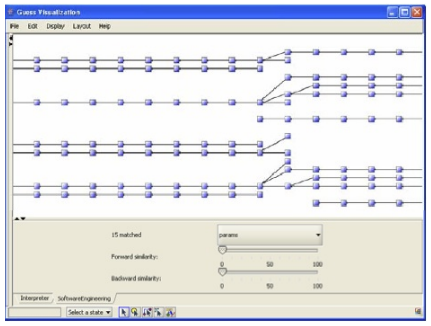

Figure 1: Genealogy Browser (from [1])

Although there are a number of tools and techniques for analyzing and visualizing clones within a single version, visualizing the evolution of clones has not gained much attention yet. So far, only Adar and Kim [1] demonstrated a tool, *Soft-GUESS*, to analyze evolutionary behaviour of code clones in software systems. They implemented three browsers to understand the clone evolution and navigate clone genealogies. The first one is the *genealogy browser* (Fig. 1) that visualizes the simple form of genealogies. The second one is the *encapsulation browser* that visualizes a tree representing the hierarchical containment of clone snippets from the snippet itself, through method, class, and package definitions. And the last one is the *dependency browser* that visualizes the genealogy graph augmented with the dependency edges. Although these views work well for small systems, they may not be suitable for large systems where thousands of genealogies are involved. In this position paper, we present an idea that could be incorporated into some existing clone visualization techniques for efficiently visualizing the evolution of clones even in large systems with thousands of genealogies.

2. OUR PROPOSAL

Since one of the primary objectives of studying the evolution of clones is to understand how clones have been changed in a given period of time, programmers or researchers usually start from the last version of the observation period to analyze clone genealogies. But certainly they could choose any reference version of interest, and could investigate which clone classes have been changed inconsistently in the past, which have remained same throughout the evolution period, how inconsistently changed clone classes are distributed in the system and so on. These questions are not simply guesswork, rather these are the fundamental questions of many research studies on clone evolution. In our previous study [5], we experienced huge difficulties to find the clone classes of interest by manually investigating all the genealogies. We can significantly offset the required manual efforts by visualizing code clones in the desired version with the information of their change patterns and could answer the aforementioned questions fairly easily. This can be achieved by adapting existing clone visualization tools and techniques for single version clone visualization with or without some minor modifications.

We explain the concept with an example of one of the most popular clone visualization techniques - enhanced scatter plot [3]. Scatter plot visualizes clones in the form of two-dimensional charts where all major software units are listed on both axes. A dot or a line segment is used in the common space of two software units if they are similar to each other. Scatter plots are useful to understand the density and relationship of clones in very large systems [4]. One can also identify all the clone pairs of a particular clone class by just clicking on any of the clone pairs of that class. We can easily exploit these advantages of the scatter plot to visualize the evolution of clones in an abstract level and then go for in-depth analysis if necessary.

Since a clone genealogy extractor provides the change history of all the clone classes in any particular version during an observation period, each clone class in that version could be visualized in the form of clone pairs using an appropriate colour based on its previous change pattern in a scatter plot. Fig. 2 shows such a scatter plot for a fictitious program having three files in a directory where function/method (*m1, m2, ..*) has been considered as a unit. Let us assume that this program has four clone classes. Among them two have been changed consistently, one has been changed inconsistently while the remaining one has not been changed at all. We use black, gray, and white colours to show inconsistently changed, consistently changed and same ('no change') genealogies respectively. Also dotted circles are indicating that those clone pairs are from same clone class. With this the users can see a more detail picture of code clones to understand their evolution. Similarly, users can see any particular class of interest such as only consistently changed genealogies, or inconsistently changed genealogies, or the genealogies in which clone fragments are added, or any pattern of interest (if the tool provides these features). Furthermore, scatter plots are useful to select and view clones, as well as zoom in/out any particular region of the plot. This facilitates users to analyze genealogies in a particular file or directory more rigorously.

While a scatter plot is useful for higher level analysis, genealogy browser (Fig. 1) could be helpful to investigate a particular genealogy. For example, let us consider the scenario where a programmer wants to see the transition point when a particular clone class has been changed inconsistently. A scatter plot can help here to find out the target

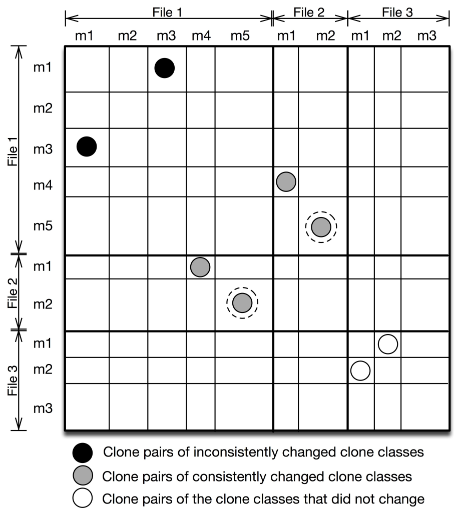

Figure 2: Scatter plot showing change patterns of clone classes

genealogy and can provide a platform to invoke the genealogy browser for investigating that found genealogy for further in-depth analysis. For example, if a user double-clicks on a particular clone pair over the scatter plot, the genealogy of its clone class could be loaded in a genealogy browser, and in this way users can find out the point where code has been actually changed. This is how a programmer can investigate clone genealogies from a very high level view to lower level details.

3. SUMMARY

Manual investigation of clone genealogies is always challenging and time consuming. In this position paper, we presented a way of visualizing large scale clone evolution that may be practical and useful both for developers and researchers. The major advantage of this approach is that existing tools and techniques for single version clone visualization can be plugged into a clone genealogy extractor [5] to visualize the evolution of clones. We hope that this paper will motivate further research in this direction and eventually we will have a suitable clone evolution visualizer.

Acknowledgements: This work is supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC).

4. REFERENCES

[1] E. Adar and M. Kim. SoftGUESS: Visualization and exploration of code clones in context. In *Proc. ICSE*, pp. 762–766, 2007.

[2] J. Harder and N. Göde. Modeling clone evolution. In *Proc. IWSC*, pp. 17–21, 2009

[3] Y. Higo, T. Kamiya, S. Kusumoto, and K. Inoue. *Code clone analysis methods for efficient software maintenance*. Tech report, SE Lab in Osaka University, 10 pp., 2004.

[4] S. Livieri, Y. Higo, M. Matushita, and K. Inoue. Very-large scale code clone analysis and visualization of open source programs using distributed CCFinder: D-CCFinder. In *Proc ICSE*, pp. 106–115, 2007.

[5] R. K. Saha, M. Asaduzzaman, M. F. Zibran, C. K. Roy, and K. A. Schneider. Evaluating code clone genealogies at release level: An empirical study. In *Proc. SCAM*, pp. 87–96, 2010.