Towards Visualizing Large Scale Evolving Clones

1st Debajyoti Mondal
Department of Computer Science
University of Saskatchewan
Saskatoon, Canada
dmondal@cs.usask.ca

2nd Manishankar Mondal
Department of Computer Science
University of Saskatchewan
Saskatoon, Canada
mshankar.mondal@usask.ca

3rd Chanchal K. Roy
Department of Computer Science
University of Saskatchewan
Saskatoon, Canada
chanchal.roy@usask.ca

4th Kevin Schneider
Department of Computer Science
University of Saskatchewan
Saskatoon, Canada
Kevin.Schneider@usask.ca

5th Shisong Wang
Department of Computer Science
University of Saskatchewan
Saskatoon, Canada
shisong.wang@usask.ca

6th Yukun Li
Department of Computer Science
University of Saskatchewan
Saskatoon, Canada
moran.li@usask.ca

***Abstract*—Software systems in this big data era are growing larger and becoming more intricate. Tracking and managing code clones in such evolving software systems are challenging tasks. To understand how clone fragments are evolving, the programmers often analyze the co-evolution of clone fragments manually to decide about refactoring, tracking, and bug removal. Such manual analysis is infeasible for a large number of clones with clones evolving over hundreds of software revisions. We propose a visual analytics framework, that leverages big data visualization techniques to manage code clones in large software systems. Our framework combines multiple information-linked zoomable views, where users can explore and analyze clones through interactive exploration in real time. We discuss several scenarios where our framework may assist developers in real-life software development and clone maintenance. Experts' reviews reveal many future potentials of our framework.**

I. Introduction

Cloning refers to the task of copying a code fragment from one place of a codebase and pasting it to some other places with or without modifications. Such clonings may give rise to several exactly or nearly similar code fragments, that form a clone class. A rich body of studies has related code clones to hidden bug propagation, late propagation, unintentional inconsistencies, and high instability. Thus software researchers suggest to manage code clones through refactoring and tracking. Clone management requires a deeper understanding of the software, and thus needs analytic tools that may answer analytical questions regarding the clones' properties, locations, distributions and co-evolution in real time.

Our primary contribution is a visualization framework, that assists programmers to visualize clone classes or communities, their distributions over the system, and co-evolution of multiple clone fragments from the same or different clone classes at a time. The goal is to enable users to intuitively explore the frequently changed clones in a software, analyze the impact of a clone change, and understand the clone co-evolutionary relations for various clone refactoring tasks.

This work is supported in part by Global Water Future (GWF) at University of Saskatchewan under a CFREF grant.

II. Visualization Framework and Implementation

Our work is inspired by the following scenarios [3]: S_1 *(Identifying clone classes for refactoring)* - how one can easily identify clones or clone classes that are important for refactoring? S_2 *(Clone tracking)* - what are the impacts of changing one clone fragment on the other related fragments from the same or different clone classes? S_3 *(Visualizing co-evolution)* - which changes that were made to a clone fragment propagated consistently over revisions? S_4 *(Analyzing clone distribution)* - which parts or modules in a software system contain most of the clones so that those parts or modules can be considered for refactoring? S_5 *(Clone usages)* - why the clones are being created? Do they capture the functionalities of the system?

1) Related Research: Adar and Kim introduced Soft-GUESS [1] to visualize clones through several browsers, where the clones have been explored as graphs or networks. CYCLONE, proposed by Harder and Göde [5], can visualize clone evolution in a matrix-like visualization, where each software version corresponds to a row and the ancestry of cloned fragments are depicted in columns. Similar visualization is available in VisCad [2]. However, none of these systems provide a visual analytics environment that supports seemless user interactions to support the clone analysis scenarios [3].

2) Design Details: Our framework is based on the principles of composite visualization [6], and the state of the art approach for visualizing big data [8]. Fig. 1(left) illustrates the main interface. The clones in (A) are depicted as scattered points and positioned using large network visualization techniques such that frequently changing clones form a few red regions. The user can select the highly changing clone fragments by surrounding them inside a rectangle by a mouse dragging operation. The selected clones may come from various clone classes. The panel (B) depicts them as a network and creates a force-directed network layout. The panel (C) shows a heatmap that depicts the pairwise coevolution frequencies of the selected clone pairs. The files network that

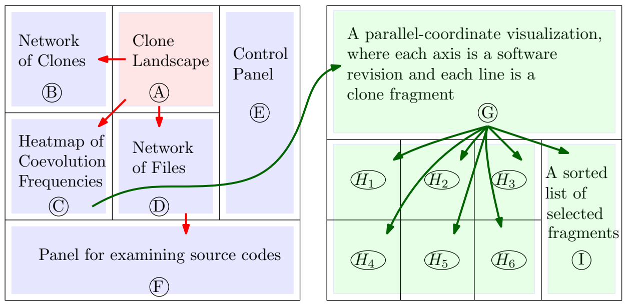
Network of Clones
B
Clone Landscape
A
Control Panel
E
Heatmap of Coevolution Frequencies
C
Network of Files
D
Panel for examining source codes
F
A parallel-coordinate visualization, where each axis is a software revision and each line is a clone fragment
G
H_1
H_2
H_3
H_4
H_5
H_6
A sorted list of selected fragments
I

Fig. 1. Illustration for the clone visualization framework.

contains the selected clones are shown in (D). The panel (E) contains parameters where the user can threshold, filter or recolor various elements as needed for the analysis. In (F), the user can examine source codes that contain the clone classes.

Fig. 1(right) helps understand the clone evolution details. The users first select the change-prone clone fragments in (C), and then press a button control to create the second interface to get the evolution details. The main panel in this interface is (G), which is a parallel-coordinate visualization for the selected clones. The panels $H_1 - H_6$ show three networks of Type 1, 2, and 3 clones, which are well-known clone types defined based on code similarities. Two clone fragments in such a network of $H_1 - H_3$ are connected, if they changed together within boundary, i.e., belong to the same clone class. $H_4 - H_6$ show the cross boundary relationships. Finally, (I) depicts a table that contains a sorted list of the selected clone fragments based on their change frequencies.

3) Implementation: We used NiCad [4] for detecting code clones and detected clone genealogies using the SPCP-Miner tool [7]. The system takes the clone and file networks in a JSON file format and does not have any data preprocessing overhead. Given the data in the appropriate JSON format, our implementation can readily visualize the dataset on its user interface.

III. Findings and Directions for Future Research

We tested our framework on two open-source subject systems, Carol and Freecol, with respectively 25K and 91K lines of code in their last revisions. The evolutionary history of these systems consists of 1700 and 1950 revisions respectively.

For each system, we analyzed the last revision available for the systems. We selected a set of highly changing clone fragments that are alive in the last revision. In each system, (S_1) a natural visual selection allowed us to choose a small set of clone fragments covering a large percentage of the clone change events. We often found clone classes that contain between 10 to 20 clone fragments, contained in a few files, which are good candidates for refactoring (due to their co-location). For the selected clones, we analyzed (S_2) the within and cross-boundary relationships among the selected clones. We noticed that the clones that have high co-change frequency, favours more within boundary relationships than the cross boundary relationships. The parallel-coordinate view revealed insights on how the clones evolve (S_3), e.g., in Carol system, we observed 12 fragments to evolve together in 4 groups. We often found small groups consisting of two to five clone fragments that changed closely over many software revisions. We also analyzed the distribution of the clones in the software systems (S_4). For Freecol, the file network view suggested high coupling among different modules. Finally, to determine the clone usages (S_5), we examined the source code files that covered the highly changing clones. This revealed many common scenarios for using clones, e.g., to encode or decode a message (url/string), construct objects or threads with different function signatures, implement conditional logic or exception handling, initialize variables, handle mouse events, prepare a string in different formats, implement a chain of function calls, and so on.

We showed the implemented system to a software development specialist from the University, and a professional software developer. They both found the system to be useful in getting a quick overview of the clones and understanding their evolution details. They also suggested to integrate code editing capabilities, such that one can use this tool for editing and save the source code, and analyze the effect in real-time. They also pointed out that code clones may originate from an author working on several files in a software package. Thus extending the framework's capability to relate clones to authors may be valuable for software maintenance and bug fixing.

In the future, we plan to address the experts' feedback and perform controlled user experiments to evaluate the implemented system. We believe that our work will inspire future research on developing interactive multiple view visualization for managing code clones.

References

[1] Eytan Adar and Miryung Kim. SoftGUESS: Visualization and exploration of code clones in context. In *Proceedings of the 29th International Conference on Software Engineering (ICSE)*, pages 762–766. IEEE, 2007.

[2] Muhammad Asaduzzaman, Chanchal K. Roy, and Kevin A. Schneider. VisCad: flexible code clone analysis support for NiCad. In *Proceeding of the 5th ICSE International Workshop on Software Clones (IWSC)*, pages 77–78. ACM, 2011.

[3] Hamid Abdul Basit, Muhammad Hammad, and Rainer Koschke. A survey on goal-oriented visualization of clone data. In *Proceedings of the 3rd IEEE Working Conference on Software Visualization (VISSOFT)*, pages 46–55. IEEE, 2015.

[4] J. R. Cordy and C. K. Roy. The nicad clone detector. In *ICPC Tool Demo*, pages 219 – 220. IEEE, 2011.

[5] Jan Harder and Nils Göde. Efficiently handling clone data: RCF and Cyclone. In *Proceedings of the 5th International Workshop on Software Clones*, pages 81–82. ACM, 2011.

[6] Waqas Javed and Niklas Elmqvist. Exploring the design space of composite visualization. In *Proceedings of the IEEE Pacific Visualization Symposium (PacificVis)*, pages 1–8. IEEE, 2012.

[7] M. Mondal, C. K. Roy, and K. A. Schneider. SPCP-Miner: A tool for mining code clones that are important for refactoring or tracking. In *Proceedings of the 22nd IEEE International Conference on Software Analysis, Evolution, and Reengineering (SANER'15)*, pages 482 – 486, 2015.

[8] Ben Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *Proceedings of the IEEE Symposium on Visual Languages*, pages 336–343. IEEE, 1996.