VisCad: Flexible Code Clone Analysis Support For NiCad

Muhammad Asaduzzaman Chanchal K. Roy Kevin A. Schneider
Department of Computer Science, University of Saskatchewan
Saskatoon, SK, Canada S7N 5C9
{md.asad, chanchal.roy, kevin.schneider}@usask.ca

ABSTRACT

Clone detector results can be better understood with tools that support visualization and facilitate in-depth analysis. In this tool demo paper we present `VisCad`, a comprehensive code clone analysis and visualization tool that provides such support for the near-miss hybrid clone detection tool, `NiCad`. Through carefully selected metrics and visualization techniques `VisCad` can guide users to explore the cloning of a system from different perspectives.

Categories and Subject Descriptors

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

General Terms

Measurement, Experimentation

Keywords

Code clones, analysis, visualization

1. INTRODUCTION

Clone detection and analysis becomes an integral part of software maintenance due to possible threats imposed by clones. Thus, over the past decade a great many clone detection tools have been proposed [4]. However, identifying important patterns of cloning requires tool support that analyzes the result and provides a higher level of abstraction with support for in-depth code-level analysis where necessary. There are also a few visualization tools proposed in the literature [4] to support such activities. However, most of them work mainly with the clone detection tools that detect only exact and renamed clones. Like several other recent clone detection tools, `NiCad` [3] has high accuracy both in terms of precision and recall in detecting exact, renamed and near-miss gapped clones [2]. Although `NiCad` gives interactive HTML output in addition to its XML textual output, the results are difficult to analyze for large code bases. To aid clone analysis with `NiCad`, we have developed `VisCad` which not only visualizes clone detection results from `NiCad` but also allows maintenance engineers to quickly locate and inspect cloning regions from higher levels of abstraction to the source code level. Although `VisCad` currently works with `NiCad`, it can be easily plugged in to any other clone detection tool that returns textual output of the detected clones.

IWSC'11, May 23, 2011, Waikiki, Honolulu, HI, USA
ACM 978-1-4503-0588-4/11/05

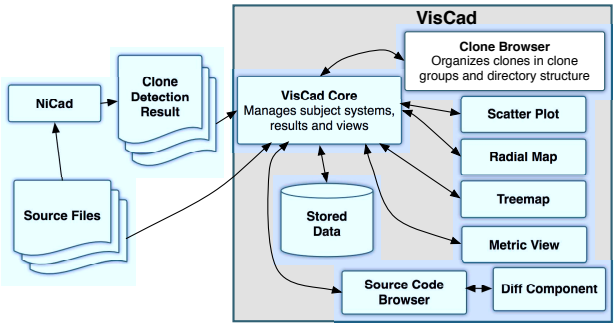

Figure 1: VisCad Architecture

In this tool demo, we will show how `VisCad` can be effectively used for large scale clone visualization and analysis with different visualization techniques and metrics.

2. VISCAD

`VisCad` works on top of `NiCad` and utilizes the results reported by `NiCad`. The architecture of `VisCad` is shown in Figure 1. Both the subject system code base and the clone detection results from `NiCad` are required as input to `VisCad`. Clones can also be detected with `VisCad's` graphical user interface. `VisCad` organizes the detected results through internal processing, and populates a set of GUI views and metrics to facilitate developers investigating the cloning status at different levels of abstraction. For instance, the source code browser (see Figure 2) allows inspection of the clones at the source code level. The changes between clone pairs can be observed using the *diff* utility integrated with the tool. In other cases where the cloning relationship is presented using different visualization techniques, the user can refer to the actual source code by interacting with the views. Since clone analysis is subjective in nature, manual removal of clones is incorporated into `VisCad` in order to provide more control to the user. At this time, `VisCad` supports scatter plot, treemap and radial map views to visually describe the cloning status of a system or a set of systems in the case of inter-project clone visualization and analysis.

2.1 Scatter Plot

`VisCad` can generate scatter plots of a subject system which is suitable for identifying cloning patterns difficult to spot in other ways. A scatter plot consists of a two dimensional matrix where each cell represents the cloning status

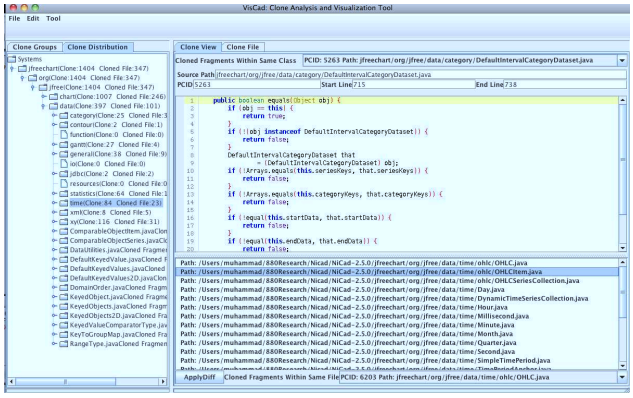

Figure 2: Source Code Browser

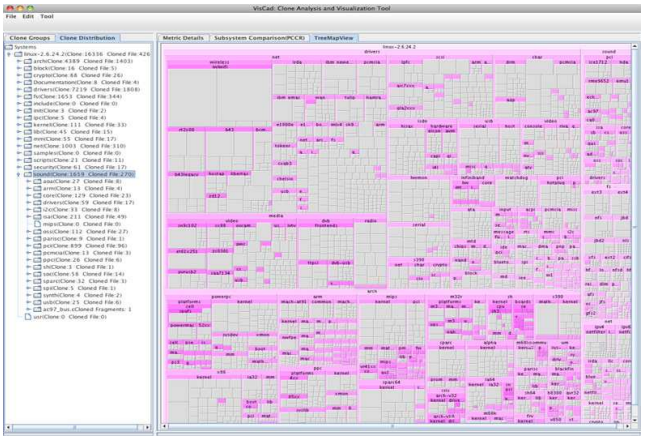

Figure 3: Tree Map View

of a pair of items. The items are labelled in the horizontal and vertical axes and can be files or directories. The level of cloning is represented using a colour heatmap. Users can have a clear abstracted view of the cloning status and can also select any cell. The source files, clone pairs or clone classes corresponding to that cell can be obtained and inspected in the source code browser. The axes size can be reduced by considering only those files or subsystems that contain clones. Moreover, a zoom in feature can help developers look deeper. A variation of scatter plot, similar to the one used by `Gemini` [5] is also available in `VisCad`.

2.2 Treemap

The treemap view shows the cloning status of directories and files through rectangles while maintaining their hierarchical structures. The size and colour of a rectangle can specify different data about the directory or file that the rectangle represents. The treemap view can be used to identify the subsystems that contribute most to the total clone pairs of a system. Often, part of a subsystem that contains the most clones may not significantly contribute to the number of cloned lines of code(LOC) in the entire system. `VisCad` allows customization of the treemap to spot locations that usually contribute to the number of cloned functions, blocks or the cloned LOC. Figure 3 shows an example of treemap created with `VisCad` for the Linux Kernel system.

2.3 Radial Map

The relationships among multiple subsystems can be explored using the radial map view, which also preserves the hierarchical structure of the subject systems. Each subsystem is represented by an arc and the length of the arc represents the level of cloning. An edge between subsystems represents the sharing of clone pairs/classes, and the thickness of an edge denotes the number of clone pairs or clone classes shared between them. It is also possible to explore such relationships for source files.

2.4 Metrics

`VisCad` populates a number of metrics including those discussed and empirically studied by Roy and Cordy [2], and supports filtering based on the metric values. For example, one particular metric named CRFM(Clone-Ratio of File for Methods) refers to the percentage of cloned methods in a given file. Source files of a system can be filtered based on the metric value to examine only those files where all methods are cloned. Such features yield opportunities for more focused investigation.

3. CONCLUSION

Using a source code browser together with different visualization techniques and metric values, `VisCad` can be effectively used for analyzing near-miss clones of large systems. Moreover, filtering or refinement of the result is also possible through `VisCad`. Thus `VisCad` meets the requirements of a clone comprehension tool that were identified by Kapser and Godfrey [1]. `VisCad` has been implemented in Java and runs on systems with Java Runtime Environment (JRE) 6. Due to a lack of space, we are unable to provide the details about the views or the metrics. However, our experience on the Linux Kernel system using `VisCad` suggests that the tool is effective for analyzing and visualizing clones even for large systems. In this tool demo, we will show all the different features of `VisCad` with the Linux Kernel as the subject system.

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

4. REFERENCES

[1] C. Kapser and M. W. Godfrey. Improved Tool Support for the Investigation of Duplication in Software. In *ICSM*, pp. 305 – 314, 2005.

[2] C. K. Roy and J. R. Cordy. Near-miss Function Clones in Open Source Software: An Empirical Study. *Journal of Software Maintenance and Evolution* 2(3): 165 – 189, 2010.

[3] C. K. Roy and J. R. Cordy. NiCad: Accurate Detection of Near-Miss Intentional Clones Using Flexible Pretty-Printing and Code Normalization. In *ICPC*, pp. 172 –181, 2008.

[4] C. K. Roy and J. R. Cordy. *A Survey on Software Clone Detection Research*. Technical Report 2007-541, 115 pp., School of Computing, Queen's University, 2007.

[5] Y. Ueda, T. Kamiya, S. Kusumoto and K. Inoue. Gemini: Maintenance support environment based on code clone analysis. In *METRICS*, pp. 67 – 76, 2002.