EgyCD Visualization for Code Clones

Ali El-Matarawy
Department of Computer Science,
Faculty of Computers and Information,
Cairo University

Mohammad El-Ramly
Department of Computer Science,
Faculty of Computers and Information,
Cairo University

Reem Bahgat
Department of Computer Science,
Faculty of Computers and Information,
Cairo University

ABSTRACT
This research presents a new visualization for code clones using EgyCD code clone detector which is based on sequential pattern mining. EgyCD presents a new graph design in which no lines has been drawn, this simplify the graph, no need for the lines since the main objective is to ease the manual management. EgyCD is independent in its visualization in which no graph tools are required for visualizing its code clones, finally supports a very nice way to ease the manual code clone management by the user. .

Keywords
Code clones, visualization, data mining, clone class, clone pairs, sequential pattern mining.

1. INTRODUCTION
The analysis of code clone data readily lends itself to graph-based analysis and visualization. However, the relationship of code clones both to themselves and other system objects creates a tangled nest of objects and links which can be difficult to explore [1].

For large software systems, clone detection tools usually report a large number (thousands) of clone pairs and clone classes. In order to help software maintainers in examining the output of clone detection tools, several clone visualization approaches and tools have been previously proposed. Previous clone visualization approaches concentrate on the following dimensions [2]

- Visualized Source Entities: Are clones shown at the code segment level, lifted to the file level, or lifted to the subsystem level. Higher abstractions (such as subsystems) permit the study of large software systems since they reduce the amount of clutter shown in the generated visualization.
- Visualized Clone Relations: Are clones shown as clone pairs, grouped as clone classes, or grouped as super clone classes. By grouping clone code segments between common files or subsystems, then software maintainers can concentrate on suspicious (large amounts of) cloning between two source entities instead of being overwhelmed by many smaller clone pairs.

Kapster et al. [3] show cloning relations in boxes-and-arrows like architectural diagrams. They visualize cloning pairs between subsystems. Figure (1) compares two different approaches (visualizing files using clone pairs and clone classes). The square nodes are files; the circle nodes are clone classes. Edges indicate cloning relationship. Both views visualize the same cloning data. The left view shows clone pairs, whereas the right view groups clones into clone classes. The left view contains more crossing edges than the right view. These edges make the visualization much harder to view in particular for large software systems. Moreover, the use of clone pairs in the visualization causes the loss of other relevant information. For example, it is not clear that the cloning relationship AB_1 is only between files A and B (clone class 2) or if it is among files A, B, C and D (clone class 1). Both of these problems are exaggerated for large software systems.

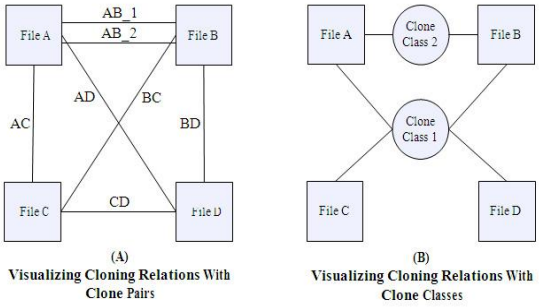

Figure 1: Comparison between two approaches [3]

2. RELATED WORK
Several visualization has been developed to illustrate the results of clone detections. In this section we present the major ones, as they are presented in [3].

The Duplication Web: is the first view that an engineer can use as it introduces the user to the duplication situation. It shows all files in the system and all existing clone connections between them. This view gives an impression of the number of files in the system and the amount of duplication that connects them. It shows the entire system at once in a well defined shape that is independent of the physical organization. It is shown in Fig. 2

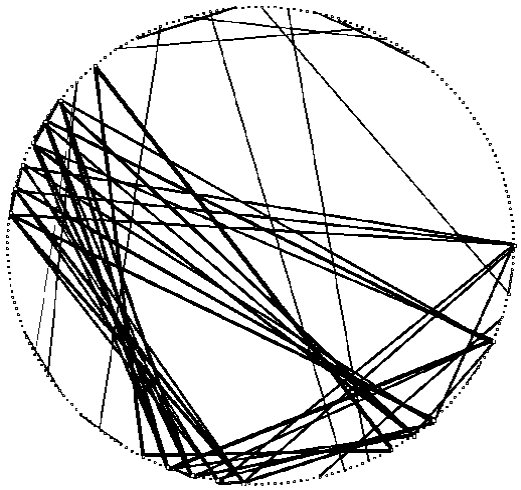

Figure 2: The Duplication Web view of MFC (Microsoft Foundation Class).

The Clone Scatterplot: it isplays the same nodes and edges as the Duplication Web but the layout takes into account the size and duplication metrics for each file. The Clone Scatterplot confronts the size of the files with the amount of duplication they contain. Files of different duplication levels can be identified by the region they are positioned in. The edges tell us if code is shared between large and small files, or between files of similar size. Heavily copied files can be selected for closer inspection. The characterstics of it is in the following:

- Nodes on the left represent small files, while the ones on the right represent large files.
- Nodes at the top of the view represent files having little or no duplication.
- Nodes that are not at the top of the view but are unconnected represent files having only internal duplication.
- Nodes close to the 45o diagonal represent files containing a lot of duplication with respect to their size.

It is shown in Fig. 3.

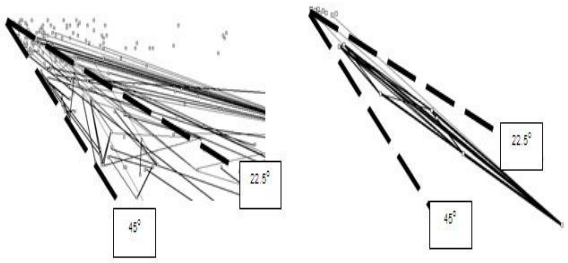

Figure 3: Two examples of the Clone Scatterplot.

The Duplication Aggregation Tree Map: it views aggregates the duplication that until now we have only seen attached to individual files. It shows the entire system top–down along the directory structure, annotating each directory node with the recursively aggregated amounts of internal and external duplication of its files and subdirectories. The view emphasizes system parts according to their involvement in duplication. The tree map aims to give an overview of the ratio of internal to external duplication, aggregated from the individual source files up to the root directory of the system. The parts of the system which exhibit high amounts of duplication can be identified at a glance from the top level. Relative comparison of structures in the hierarchy is made possible. The view has a gestalt property, i.e., it can give useful information immediately. It is shown in Fig. 4.

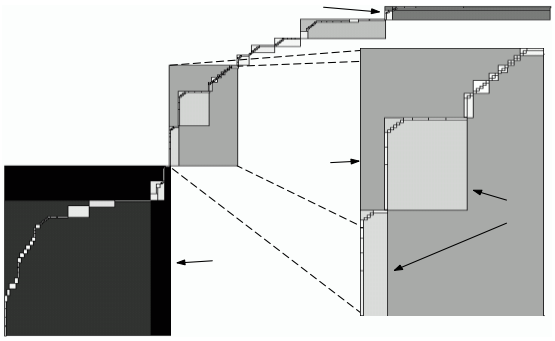

Figure 7.4 The tree map of the APACHE system.

The Clone Class Family Enumeration: it views reduces the redundancy of the duplication connections that has been presented in all the previous views. The clones are shown in a view of concise nodes and edge. The layout uses the LCC (Lines of Copied Code) and the LOC (Lines of Code) metrics to place clone class families and source files, respectively, on the horizontal axis. The intuition "the farther to the right the bigger" thus can be used to mentally classify both entity types presented in the view. The edges connect the clone class families in the upper half of the view with the source files on the lower half. This view presents the clone class families to the user in a way that eases investigation of individual instances of duplication. It characterizes the families by the criterion of how many source files they comprise and how much code they contain. The user can start on a clone class family node and see which source files are participating. He can also start with a source file node and see in how many clone class families the file participates. To make the view fully useful, lower level duplication entities, i.e., clone classes and finally clones must be made available to the user via the nodes in this view. It is shown in Fig. 5.

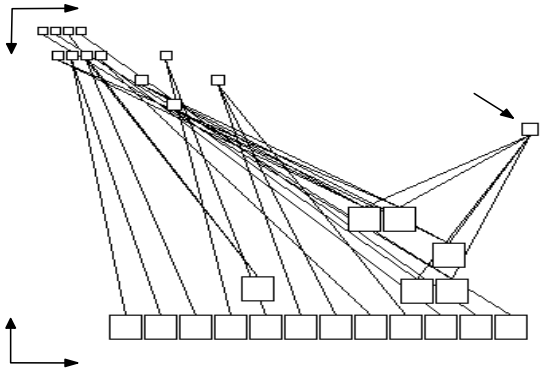

Figure 5: The Clone Class Family Enumeration

3. VISUALIZATION GRAPH TOOLS

Most code clones detectors use one of two graph tools to present the code clones namely:

3.1 AJDT Visualizer

The AJDT Visualizer is an Eclipse plugin that is part of the AspectJ Development Tools project. The developers of the Visualizer opened the plugin for adaptation by providing several extension points to allow other types of information to utilize its visualization features [4].

3.2 GUESS system

The GUESS system provides the users with a mechanism to interactively explore graph structures both through direct manipulation as well as a domain-specific language [1].

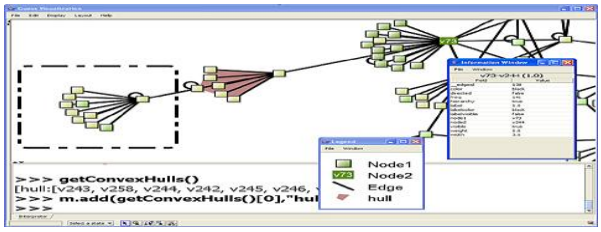

Figure 6: A screenshot of the GUESS system [1].

When we decided to visualize code clones as graphs in EgyCD [5,6] we wanted to keep the easy way in code clone management for the user and if we switched to an external-graph tool we may not achieve the target of maintaining the

source code by the developer so we decided to build the graph in EgyCD instead of using of a graph tool.

4. EGYCD VISUALIZATION

Most visualizations tools abstract and display cloning information at the subsystem level but with different emphases. Most of them participate in using lines or edges to represent relationships between code clones and files or folders. In our opinion too many lines to be displayed will make the graph too difficult to understand. Therefore we decided to visualize code clones as shown in Appendix (B) Screenshot (B.7), in which all code clone classes are listed in a master form and all of its locations data are detailed in sub-form beneath it. The user can double click any row in the detailed form and the file contains the code clone will be opened. In this way, an easy manual management can be done by the developers of the source code to maintain it. This was our focus but at the same time we lost the most important feature of the graph which is showing everything in one sheet, so it means knowing a lot of information in a second just with one look to the graph. This feature comes from the graph's nature either it is for code clones or for something else such as line graph for cities, building projects,…,etc.

Graph views achieve the goal of data reduction on different levels. We are able to display even very large systems on restricted screen space. Many of the views have a gestalt property, i.e., they provide overview information at a glance. The reduction of the cardinality of the clone sets, however, is sometimes not enough, resulting in cluttered displays which are hard to read. We must further support readability with interactive enhancements of the views, e.g., with the highlighting of connected elements on mouse over, mouse click or double click.

By using simple and heuristic layout mechanisms, we provided a fixed arrangement of the nodes representing them as code clone objects of the code clone classes as shown in Fig. 7. This is an advantage since there is no need for the user to rearrange the nodes to get a better view, also no need for the lines since the main objective is to ease the manual management; the user can click any node to know all information about the class of the node and if the user wants to start the manual maintenance to his source code he can click the node then EgyCD will open it automatically

4.1 EgyCD Graph Design

We sort all code clones classes on their repetition in descending order then we draw them in a virtual rectangle, in which each group of adjacent nodes represent one class and each node in this group represent a clone class in a file containing this clone class. To differentiate among these clone classes we draw each clone class with a specific color. Instead of having one big virtual rectangle containing all data, we divide it to smaller virtual rectangles, this will improve only the readability of the graph as shown in Fig. 7.8.

We called each virtual rectangle a code clone item in which it contains different instances (objects) of code clones classes. Each code clone item has a fixed width with a primary fixed height and it can be increased for adding extra nodes of a class, so we don't allow objects of the code clone class to be divided in two virtual rectangles.

In Fig 6, the first code clone item contains the code clone class with high counts since as being mentioned earlier, we sort them in descending orders and also this can be noticed in the last column in which all code classes almost has a repetition count equal 2.

Code Clone Item

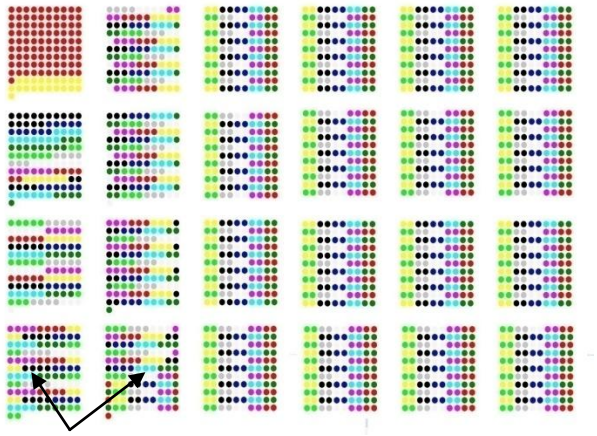

Figure 7: EgyCD graph representation

4.2 EgyCD Graphical Interface

When the user clicks on any node, a full data (code clone folder location, the file it contains, the first line of the code clone, and the code clone size) is displayed as shown in Fig. 8. (the displayed clone data displayed in that figure can be obtained by clicking any of the nodes under the shadow rectangle of second clone item in the first row) and when the user double clicks the node the corresponding file of that node is opened automatically for maintenance or management manual process by the developer of the source code.

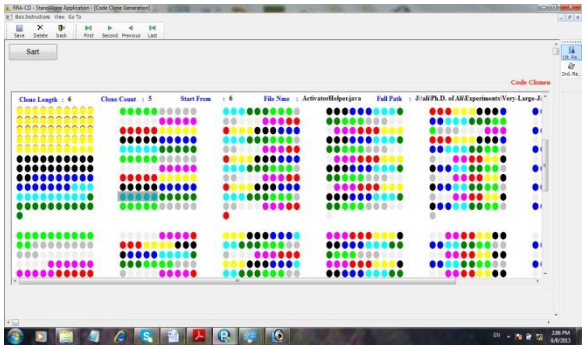

Figure 7.7: EgyCD Data representaion

4.3 EgyCD Graph features

Generally the visualization of code clones has three features, the first feature shows the locations of the code clones, the second feature shows which code clones are important than other code clones, and the third feature is the way of representations. EgyCD graph supports all of these features in simple and interactive way for source code manual management.

EgyCD grah has the following features:

- EgyCD supports all features of code clones visualization.
- EgyCD presents a new way of representation in which it has no edges.
- EgyCD supports class representation, it is so clear from Fig. (7.1) that class representation is more clearer than clone pair representation.
- EgyCD is independent in its visualization in which no graph tools are required for visualizing its code clones.
- EgyCD supports a very nice way to ease the manual code clone management by the user.

5. CONCLUSIONS

In this paper, we presented new graph design for visualizing code clones in a simple and powerful features. EgyCD visualizations has n lines and hence no time is needed to understand all the information it presents. EgyCD supports the class visualization. EgyCD is independent in its visualization in which no graph tools are required for visualizing its code clones, also it supports a very nice way to ease the manual code clone management by the user.

6. REFERENCES

[1] E. Adar and M. Kim, SoftGUESS: "Visualization and Exploration of Code Clones in Context", University of Washington, Computer Science and Engineering, Software Engineering, 2007. ICSE 2007. 29th International Conference.

[2] Z. Ming Jiang, "Visualizing and Understanding Code Duplication in Large Software Systems", A thesis presented to the University of Waterloo in fulfillment of the thesis requirement for the degree of Master of Mathematics in Computer Science Waterloo, Ontario, Canada, 2006.

[3] C. J. Kapser and M. W. Godfrey. "Supporting the Analysis of Clones in Software Systems: A Case Study", Journal of Software Maintenance and Evolution: Research and Practice, 18(2), 2006.P. Clough ,"Plagiarism in natural and programming languages: an overview of current tools and technologies", July 2000, Department of Computer Science, University of Sheffield.

[4] R. Tairas, j. Gray and I. Baxter, , "Visualization of Clone Detection Results", In Proceeding eclipse '06 Proceedings of the 2006 OOPSLA workshop on eclipse technology eXchange , 2006, Pages 50 – 54.

[5] A. Matarawy, M. El-Ramly and R. Bahgat. " Plagiarism Detection using Sequential Pattern Mining, International Journal of Applied Information Systems (IJAIS) – ISSN : 2249-0868 Foundation of Computer Science FCS, New York, USA Volume 5–No.2, January 2013 – www.ijais.or , pp 24-29.

[6] A. Matarawy, M. El-Ramly and R. Bahgat. " Plagiarism Detection using Sequential Pattern Mining, International Journal of Applied Information Systems (IJAIS) – ISSN : 2249-0868 Foundation of Computer Science FCS, New York, USA Volume 5–No.2, January 2013 – www.ijais.or , pp 24-29.

[7] A. Matarawy, M. El-Ramly and R. Bahgat. "Parallel and Distributed Code Clone Detection using Sequential Pattern Mining", International Journal of Computer Applications (0975 – 8887) Volume 62– No.10, January 2013, pp 25-31.