Visualization of Software Code Clones-A Review

Sandeep Bal
Research Scholar, PTU

Sumesh Sood
Department of Computer Science,
Swami Satyanand College of Management & Technology, Amritsar

Abstract- Code cloning practices are a common trend in software development phase. It has both good and bad perspectives. If it is the need of the hour, cloning is a boon and if the maintenance of the software is the issue then cloning is a bane as it leads to bad software architecture due to copied and modified code. It leads to increased complexity or a degraded overall architecture of the software. Such issues arise during the maintenance/re-engineering activity. The refactoring of the clones is required in order to maintain the software architecture and stability. The visualization of the software cloning is required to decide the most appropriate action for code clone maintainability. In this research paper, various visualization techniques are briefly reviewed and the working has been explained.

Keywords: Code clones, Software, Maintenance, Refactoring, Visualization

I. INTRODUCTION

Software maintenance is an important aspect of software re engineering. It should be well designed and performed with utmost care. The code should be well organized so that the software architecture does not suffer in case of code cloning activity.

The main purpose of the visualization is to highlight the cloning within each subsystem and across subsystems to the developers. And hence, they can then easily investigate whether the cloning is justifiable or not.

Another goal of visualization can be to show the dependency of different subsystems on each other.

Visualization allows users to analyze the evolution of clones from a coarse grain to a fine grain level. The concept of visual exploration elaborates the source code analysis for the user to predict and see the changes to clones (if they occur) in a future release/ revision of the software along with the proper handling of a large volume of clone data.

In this paper, various visualization techniques have been briefly reviewed. A work brief has been written along with the tools used by the techniques. It was studied that the visualization has been categorized on the basis of clone entities and their relationship at levels such as clone pair level, clone classes or super clones.

II. VISUALIZATION TECHNIQUES- WORK BRIEF

2.1. A Hasse diagram[1] uses directed acyclic graph to present the partial order. The nodes that are related by the order relation are connected by Directed arcs and for which no other directed path exists. The directed acyclic graph is then shown with all arcs pointed down (or all pointed up) so that arrow heads are not necessary to indicate the direction.

A common partial order involves a collection of subsets with inclusion as the order relation. In the case described here, a range of characters (or lines) from two (or more) files having similarity in a block of text is identified. The characters from each subset of file can be counted and matched between the given set of files. Only nodes with non-zero matches will be considered.

A special-purpose Icon program is used to aggregate the matches for subsets of files and calculate the Hasse diagram. The results were then output in GraphEd format and this graph editor manually organizes the graphs for presentation.

A vertical co-ordinate related to the size of the file or match between a set of file is involved in the visualization. Then matches that are larger in absolute or relative terms stand out without totally hiding weaker matches.

The vertical placement of nodes vertically given by the logarithm of size in Hasse diagrams makes them a useful technique for visualizing interactions in complex clusters of matches. In particular, the several causes of matches present in the 18-file cluster are relatively easily separated by this analysis.

This powerful method of exact matching of sequences of lines provides much information with comparatively little noise. The complexity present when more than one kind of matching is involved can be addressed using Hasse diagrams.

2.2. In this technique[2], the technology of HTML and the World Wide Web was taken as the primary issue and a tool for data visualization and navigation of the textual redundancy web was designed. All the various entity types are mapped to HTML pages. Prior to that, an introduction to underlying technology of Analysis of Redundancy in Text[1,3,4,5,6] and its normalized data model is performed. Each entity represents an entity instance and is connected to its neighbours by hypertext links. De normalization and a layout is considered essential in order to improve the usefulness of the resulting pages. The success of this prototype can be attributed to a coming together of well-established ideas from the areas of database design and hypertext document preparation with the new web based technologies.

The legacy systems contain repeated text in the form of large and small blocks that appear in more or less the same form in several places due to the continuously performed maintenance activity. Such repeated data blocks helps in defining a structure that can contribute information about the development history of the source that is different from the documented version or the current directory structure. The result is a network of linkages containing useful information is found when such textual matches are brought together. The repeated fragments of text are the result of the languages used, the design process, the use of common idioms, and maintenance activity. A navigation path is formed by such linkages, and the overall structure can be inferred by navigating them. A basic data model is designed then to synthesize all the low-level observations which helps to summarize the understanding gleaned and proves out to be consistent with structure derived from other sources.

The Basic Data Model

There are six basic entity types: File, Directory, Snip, Hash, Cluster and Component. There are a number of relationships between these entities:

- File or Directory to parent Directory.
- Snip to the File containing it.
- Snip to Hash value.
- File to its singleton Cluster.
- Hash to the Cluster implied by its match set.
- Cluster to the Component containing it.
- Component to the Cluster containing exactly its set of files.
- Cluster to Cluster containment relation.

All of these are found to be functional relationships except the last. The containment relationship of clusters is more complex to represent. A cluster may contain many smaller clusters, and may be contained in many larger clusters. The focus has been given on cases where one cluster is contained in a second, but there are no other identified clusters that are properly between the two. Since it helps to reduce the number of linkages to be displayed on pages and navigate the clusters in a number of steps. The idea was exactly same while defining Hasse diagrams of clusters [1,6].

The reports containing information about these entities and relationships are produced by the analysis of the matches. These reports use the keys like File ID, Directory ID, Snip ID, Cluster ID, Hash ID and Component ID. These entire IDs' are preceded by their type in order to reduce the confusion.

After designing the model, the solution to design problem must be made with the following points in consideration:

- For a given file, show how it is composed of snips in a way that the matches can be easily discovered.
- For a given file, find other files that have matches with it.
- Find interesting clusters of files that contain matches with useful information.
- Get information similar to that provided by the "diff" tool but generalized to more than two files, or to files whose matching pieces occur in different orders.

The most obvious common thread among the above is that most of the emphasis is on files, file contents, clusters, and how clusters relate to files and their contents. The design should focus on these. The model is denormalized, in order to simplify the navigation between files, clusters, and components, is handled in the fully normalized model.

All such details were shown by taking an example of a recent version of the GCC Compiler. As mentioned before [1,3,4], this has the advantage of being generally available in source form, being of appropriate size, and having sufficient redundancy as to provide a useful exercise.

The conclusion made was that the user must be involved in the design. The usability of the system depends very much on how well it helps users solve problems. It is up to the user to make that determination. It was concluded that it is very easy to work in a medium if one is clear about the data model and how it should be denormalized.

2.3. This paper[7] proposed a way of grouping the duplication information into useful abstractions, *i.e.,* the clone class family which aggregates all duplication that is exchanged between a specific set of files. It also proposed a number of polymetric views which structure the data and combine it with the knowledge about the system that the engineer already possesses.

A Graph is a natural way of expressing relations where the nodes of the graph represent the source entities whereas the edges of the graph represent the duplication relations. Any duplicated data is relational data where two source code entities are related by shared pieces of code. According to Ware [8], visualization is the preferred way of getting acquainted with and navigating large data pools and Graphs are considered one way for Clone Visualization. Such investigation must consists of the following elements:

The source entities represent (fragments of) the source code. Files were used as source entities. Other entities such as subsystems, modules, classes, and methods could also be used as well.

The duplication relationships connect the source entities. The duplicated fragments are the source code that two (or more) source entities have in common. Thousands of clones could be found due to the millions of LOC. The related clones were aggregated into higher level entities in order to achieve scalability because the visualization of all individual clones lead to hinder the interpretation due to over plotting of nodes and edges. Three duplication entities were defined which formed a containment hierarchy. Each higher order entity aggregates lower level entities. Following duplication entities were identified:

1. Clone Pairs: The lowest level of detail on which to describe duplication is the *clone pair* <a, b>. The pair comprises two source code fragments a and b which are copies of each other.

2. Clone Classes: A clone class is the union of all clone pairs which have source fragments in common. For example, if we have the clone pairs <a, b> and <b, c>, it is likely that there is a clone pair <a, c>. The *clone class* then encompasses the fragments a, b, c. The *domain* of a clone class is the set of source entities from which its source fragments stem.

3. Clone Class Families: A *clone class family* is formed by grouping all clone classes that have the same domain.

It was noted that the clone class family was the topic of interest for the reengineer: First, since clone class families contain only entire clone classes, they assemble all instances of a source fragment found in the system. Second, a clone class family reveals duplication activity that goes beyond the duplication of a single continuous source fragment. If two fragments, which were initially copies of each other, evolve differently over time, they may not be recognized as one clone pair any more (the “split duplicates” problem mentioned in [9]). The clone detector may identify smaller parts which are still similar individually. A clone class family will reunite those clone fragments.

It was summarized that a clone class family aggregates all elements that are necessary to make informed decisions about refactoring measures for a particular fragment of copied code. The proposed visualizations used the source files and the clone class families as entities.

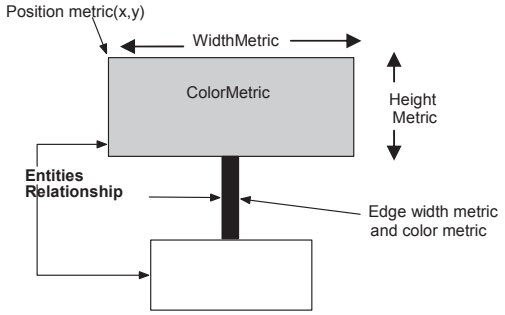

Figure 1

The principle of Polymetric Views

Polymetric views [10] are a visualization method for nodes–and–edges graphs enriched with semantic information such as metrics. Figure 1 illustrates how two-dimensional nodes representing entities, ***e.g.,*** software artifacts, and edges representing relationships can be enriched by software metrics: A node figure is able to render up to five metric values: in its width, height, x– and y–position, and in its color. An edge figure is able to visualize two metric values: width and color. By applying metrics to the x– and y–position of the nodes, for example, similar entities are clustered close together in an easily identifiable region of the graph exhibiting some of their defining characteristics. Entities with differing characteristics are then placed in a distinct region of the graph. In this way, the shape of the visualized graph is able to communicate useful facts about the set of all visualized items.

Duplication Metrics

A number of metrics that characterized the source files and clone class families were selected to discern between instances of code duplication. The metrics were found to be simple and could be computed from the results of any duplication detection tool without the aid of a parser.

Polymetric Views of Duplication

A set of polymetric views were proposed that support the understanding of the duplication situation in a system and can -guide refactoring measures. The description of the views were ordered in a sequence suggesting a way for the engineer to walk through the task of understanding a system's duplication (a *reengineering roadmap*). Each view was presented using the following schema:

Details. Gives a tabular technical description of the view, its entities, relations, and its layout.
Intention. Explains how the view can support the engineer in his tasks.
Symptoms. Details what kind of duplication problems the view reveals.
Examples. Shows sample views and explains their features.
Scaling. Investigates how the size of a system affects the view negatively and what can be done about it.
Overplotting. Investigates if the amount of data can cause overplotting problems and how they can be avoided.

A short overview of questions answered and potential further questions were presented after the discussion of each of the following views.

- The Duplication Web.
- The Clone Scatterplot.
- The Duplication Aggregation Tree Map.
- The System Model View.
- The Clone Class Family Enumeration.

It was concluded that the views achieved the goal of data reduction on different levels. Many of the views were found with a *gestalt* property, ***i.e.,*** they provide overview information *at a glance*.

A fixed arrangement of the nodes for all views except the System Model view was presented by using simple and heuristic layout mechanisms. This approach discussed put an emphasis on the *'human in the loop'*, and gave human expertise the helm, instead of pushing automatization.

2.4. Live Scatterplots [11] are aimed at providing an immediate, intuitive answer that can help the analyst to quickly identify and access subsystems and clones involved in a pattern simply by directly pointing at it in the Scatterplot.

Scatterplots as Web Pages

Live Scatterplots is a tool that accepts the output of any clone detector and automatically renders the clone pairs as an interactive web page that displays clones as a scatterplot over the subsystems (directory structure) of the original source system or systems. The axes are specified as two files of directory names to be used for the rows and columns of the plot respectively. These files specify the level of abstraction to be used. They can be source

file names (for small systems), low level source directories (for mid-sized systems) or higher-level subsystem directories (for very large systems). The axes can use the same set of directory names (if the plot is of clones in a single system) or two different sets (for clones between two systems).
The implementation of scatterplots as web pages was found very simple. The tool simply examined each clone pair reported by the clone detector, pattern matched the first fragment's file path with the directories of the rows and the second with the directories of the columns, and formed a matrix with the pairs in the corresponding cells. When all pairs were placed in the matrix, Live Scatterplots rendered the matrix as an HTML table, by using cell colour to represent clone density, yielding a web page.
Live Scatterplots can display either the entire matrix, or can reduce the axes to include only those directories which actually contain clones. In either case, rendering is fast and accurate in modern browsers. Live Scatterplots can render the table with either a black background, to more easily see low density cloning, or a white background, to better see very dense plots, and with or without labels.

2.5. The clone detection results were visualized[12] by extending the Visualiser plugin that is part of the AspectJ Development Tools (AJDT) project. The close connection between aspects and clones made this Extension of the Visualiser possible for displaying clone detection results[13]. Both aspects and clones share similar characteristics in representing code that is duplicated and scattered throughout the source code of a program.
The plugin connects the CloneDR tool with a visualization feature that is an extension of the AJDT Visualiser. Additionally, the integration of CloneDR with Eclipse allows the tool to take advantage of the rich environment of the IDE, which offers frameworks for wizards, views, and editor connections.
The clone visualization described in this paper compliments the standard text listing and scatterplot visualization and can help users to identify certain characteristics of the clones by graphically isolating clone groups across a list of source files. With the addition of the Visualiser view, the CloneDR tool has been enhanced to provide a graphical interface through Eclipse integration.The Visualiser plugin was extended through one of the provided extension points, but a feature that was needed required modification of the Visualiser source code. The future work would allow the additional features that would assist users to understand the clone detection results and enhance the clone detection process. Few points were as listed below:
• *A more structured view of the source code files*: If the language is object-oriented, the classes could be displayed with respect to their relationships in a manner similar to a UML class diagram. This display could reveal clones such as those that are duplicates in subclasses, which could be addressed through the Pull-Up Method refactoring.
• *Incorporating more information from the results file*: The results file produced by CloneDR contains additional information about the detection process that includes the timing of most procedures and parameter bindings of each clone. These are not currently displayed in our plugin, but will be incorporated in future versions.
• *Tighter integration of CloneDR in Eclipse*: A mechanism that can allow CloneDR to be integrated further into Eclipse could simplify and speed up the detection and visualization process by providing the Visualiser direct access to the CloneDR internal representation.

2.6. The concept of clone mining was introduced in this[14] research work. **Clone mining** is defined as the process of uncovering interesting clones and clone patterns from the large amount of clone candidates produced by clone detection tools.
This framework was influenced by traditional data mining frameworks [15, 16] which focuses on uncovering previously unknown or potentially useful clone patterns in large sets of clone data whereas traditional data mining frameworks focus on extracting previously unknown or potentially useful information from large data sets. It used data reduction and visualization techniques in which the clone data is reduced by aggregating clone information at various abstraction levels and removing irrelevant data. A visualization tool was offered which allows users to query and explore clone data at various abstraction levels.

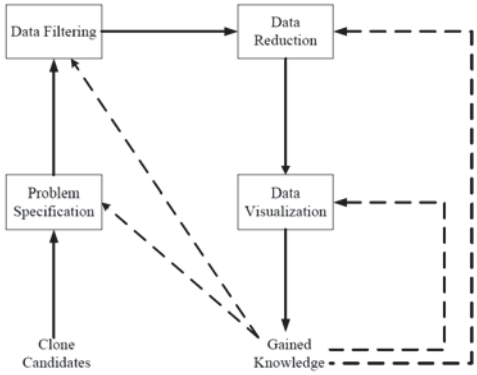

Figure 2

An Overview

The figure consists of four steps**:** Problem Specification, Data Filtering, Data Reduction and Data Visualization. **Clone Candidates** are segments of code which were reported as clones by clone detection tools. The various clone detection tool uses different heuristics to decide if two code segments are similar (i.e., clone candidates). For example, in metric based clone detection tools [17], code segments which share similar metric value are considered as clones. In abstract syntax tree (AST) based clone detection tools [18], code segments which have the similar AST subtrees are considered as clones. No technique is perfect, as these techniques might miss clones or report false positive clones [19].

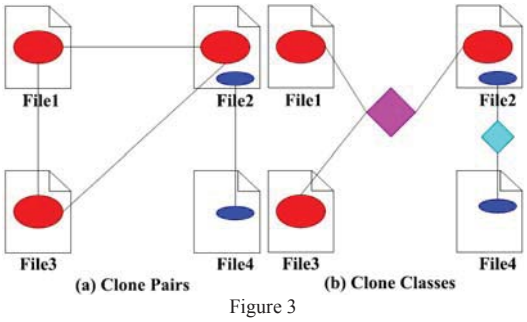

Figure 3

The reported clone candidates were represented as either clone pairs or clone classes. A clone pair was defined as a pair of code segments which are identical or similar to each other. A clone class was defined as the maximum set of code segments in which any two of the code segments in the set form a clone pair. For example Figure 3(a) shows 4 clone pairs. The three red code blocks in "File1", "File2", and "File3" form three clone pairs. The two blue blocks in "File2" and "File4" form another clone pair. Figure 3(b) represents the same clone information using clone classes instead of clone pairs as shown in Figure 3(a). Figure 3(b) has two clone classes. A pink diamond indicates a clone class connecting the three red blocks from "File1", "File2" and "File3". Another blue diamond represents another clone class connecting the two blue blocks from "File2" and "File4".

Problem Specification It described the purpose of the analysis where *interesting* clones were defined as that one should look for (i.e., mine) and not interesting clones which one should prune. The problem specification also studied the spread of the clones which is the scattering of clones across a software system. Clone pairs or classes may contain files that reside within one subsystem (internal clones) or files which reside in many subsystems (external clones).

The technique of Data Filtering was used to remove false positives clones i.e. the operation of removing noise from the data. For example, the CCFinder tool uses a "Parameterized Token Matching" [20] heuristic to decide whether two code segments are similar. All the reported clones by CCFinder exhibit similar structures. However, the clone candidates may not share similar semantics.

Textual filtering technique was considered suitable for improving the precision of the clone data where pairs of code segments are compared and a textual similarity between them is calculated using the diff algorithm. Taking the example of two code segments A and B, the textual similarity between A and B would be defined as T(A, B) = s/(a+b –s), where *a* is total number of lines in code segment A; *b* is total number of lines in code segment B; and *s* is lines of code in which A and B are similar. If A and B are identical, then T(A, B) is 1. If A and B have nothing in common, then T(A, B) is 0. If the textual similarity is above a threshold, then these two code segments are considered as a clone pair. Otherwise, these two code segments do not form a clone pair and should be removed from the clone data set. The filtering threshold is determined by trial-and-error in an iterative fashion. This research started with a threshold of 0.01 and explored the results of the filtering using a random

sampling of 100 clone candidate pairs. The threshold was incremented by 0.01 at each step on the basis of the results of the random sampling, until the user of the framework was satisfied with the precision of the data.
Data Reduction is concerned with systematically reducing the volume of data. Two data reduction approaches were considered: aggregation and pruning.
Data aggregation abstracts the clone data to highlight general trends. Data pruning removes the irrelevant clone data to make interesting clone information more visible. Two data aggregation approaches were introduced here: merging and lifting.
The merging approach combines clone classes with different line ranges in the same file into a single clone. The lifting approach abstracts clone relations from the code segment level up to the file level or up to the subsystem (i.e., directory) level.
Data Visualization displays the processed data in order to summarize and highlight interesting clones. The researchers have developed various visualization techniques. E.g. **Clone Cohesion and Coupling (CCC)** graph and **Clone System Hierarchical** (CSH) graph. The CCC graph highlights the amount of internal clones (clones within subsystems) and external clones (clones across-subsystems) for a single level of system abstraction. The **Clone System Hierarchical** (CSH) graph provides an interactive mechanism for exploring and querying the clone data. The visualization uses a tree layout which mimics the directory structure of the File system. The tree layout enabled the display of data at different levels of abstractions instead of having to lift the data to a particular abstraction level (as done in the CCC graph). The tree layout showed clones at all levels of abstraction in the same time.
Gained Knowledge refers to interesting clone patterns or clone phenomena uncovered from the clone candidates.
Knowledge discovery is an iterative process as shown by the dotted lines in Figure 2. The **Feedback** loop can refine any of the defined framework steps or specify new problems. The whole concept was explained by taking a case study of "The Linux Kernel".

III.SUMMARY

The work of various researchers has been briefly reviewed by highlighting the basic working scheme of each visualization technique.

REFERENCES

[1] J. H. Johnson, "Visualizing Textual Redundancy in Legacy Source", *Proceedings of the 1994 CAS Conference,* pages 9–18 (October 31–November 3, 1994)
[2] J. H. Johnson, "Navigating the textual redundancy web in legacy source. "*Proceedings of the 1996 conference of the Centre for Advanced Studies on Collaborative research*. IBM Press, 1996.
[3] J. H. Johnson, "Identifying Redundancy in Source Code using Fingerprints", *Proceedings of the 1993 CAS Conference,* pages 171–183 (October 24–28, 1993).
[4] J. H. Johnson, "Substring Matching for Clone Detection and Change Tracking", *Proceedings of the 1994 International Conference on Software Maintenance,* (September 19–23, 1994).
[5] J. H. Johnson, "Using Textual Redundancy to Understand Change", *Proceedings of the 1995 CAS Conference,* CD-ROM (November 7–9, 1995).
[6] M. Whitney, K. Kontogiannis, J. H. Johnson, B. Corrie, E. Merlo, J. G. McDaniel, R. De Mori, H. A. Müller, J. Mylopoulos, M. Stanley, S. R. Tilley, and K. Wong, "Using an Integrated Toolset for Program Understanding", *Proceedings of the 1995 CAS Conference,* pages 262–274 (November 7–9, 1995).
[7] R. Matthias, Stéphane Ducasse, and Michele Lanza., "Insights into system-wide code duplication" *Reverse Engineering, 2004 Proceeding.. 11th Working Conference on*. IEEE, 2004.
[8] C. Ware. *Information Visualization*. Morgan Kaufmann, 2000.
[9] J. Krinke. Identifying similar code with program dependence graphs. In *Proceedings WCRE'01*. IEEE Computer Society, Oct. 2001.
[10] M. Lanza and S. Ducasse. Polymetric views - a lightweight visual approach to reverse engineering. *IEEE TSE*, 29(9):782–795, Sept. 2003.
[11] Cordy, James R. "Live scatterplots." *Proceedings of the 5th International Workshop on Software Clones*. ACM, 2011.
[12] Tairas, Robert, Jeff Gray, and Ira Baxter. "Visualization of clone detection results." *Proceedings of the 2006 OOPSLA workshop on eclipse technology eXchange*. ACM, 2006.
[13] Bruntink, M., van Deursen, A., van Engelen, R., and Tourwé, T. On the Use of Clone Detection for Identifying Crosscutting Concern Code, *IEEE Transactions on Software Engineering*, vol. 31, no. 10, October 2005, pp. 804-818.
[14] Jiang, Zhen Ming, and Ahmed E. Hassan. "A framework for studying clones in large software systems." *Source Code Analysis and Manipulation, 2007. SCAM 2007. Seventh IEEE International Working Conference on*. IEEE, 2007.
[15] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. Introduction to Data Mining. Addison-Wesley, 2005.
[16] Peggy Wright. 1998. Knowledge discovery in databases: tools and techniques. Crossroads 5, 2 (Nov. 1998), 23-26.
[17] Kostas Kontogiannis. Evaluation Experiments on the Detection of Programming Patterns Using Software Metrics. In Proceedings of the Fourth Working Conference on Reverse Engineering, 1997.
[18] I. D. Baxter, A. Yahin, L. Moura, M. Sant'Anna and L. Bier. Clone detection using abstract syntax trees. In *Proceedings of the International Conference on Software Maintenance*, page 368,Washington, DC, USA, 1998. IEEE Computer Society.
[19] E. Burd and J. Bailey. Evaluating clone detection tools for use during preventative maintenance. In Second IEEE International Workshop on Source Code Analysis and Manipulation, pages 36–43, 2002.
[20] Toshihiro Kamiya, Shinji Kusumoto and Katsuro Inoue. CCFinder: A Multi-Linguistic Token-based Code cloning Detection System for Large Scale Source Code. IEEE Transactions on Software Engineering, 28(7):654–670, July 2002.