Visualization of Clone Detection Results

Robert Tairas and Jeff Gray
Department of Computer and Information Sciences
University of Alabama at Birmingham
Birmingham, AL 35294-1170
1-205-934-2213
{tairasr, gray}@cis.uab.edu

Ira Baxter
Semantic Designs, Inc.
12636 Research Blvd., C214
Austin, TX 78759-2239
1-512-250-1018
idbaxter@semanticdesigns.com

ABSTRACT
The goal of a clone detection tool is to identify sections of code that are duplicated in a program. The result of the detection is presented in some manner for the user to view, which is usually in the form of a list of clones that are grouped together. Previous research has shown how scatter plots can be used to render a graphical representation of the results. This paper describes the integration of a stand-alone clone detection tool into Eclipse and a corresponding alternative visualization of clone detection results. An Eclipse plugin is described that displays the results of a clone detection tool called CloneDR™. The visualization of the results is implemented as an extension to the AspectJ Development Tool (AJDT) Visualiser plugin, which is primarily used to view crosscutting concerns in aspect-oriented programs.

Categories and Subject Descriptors
D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement – *restructuring, reverse engineering, and reengineering*.

General Terms
Management, Design, Economics, Human Factors, Languages.

Keywords
clone detection, visualization.

1. INTRODUCTION
Code clones are sections of code that occur in multiple locations in a program. Clone detection tools aim to automatically search for clones and report any detected clones back to the user. A textual representation of the result consists of clones being listed together, along with the source file names and line locations (i.e., starting and ending line location) of each clone instance. A scatter plot is a popular graphical representation of clone detection results where duplicate sections of code are identified as a sequence of connected dots in a graph.

This paper describes a new approach to visualization of clone detection results that takes advantage of the ability to extend the Visualiser plugin that is part of the AspectJ Development Tools (AJDT) project[1]. The Visualiser itself is primarily used to display the location of crosscutting concerns in an aspect-oriented program. Extending the Visualiser to display clone detection results becomes obvious because of the close connection between aspects and clones [2]. Both aspects and clones share similar characteristics in representing code that is duplicated and scattered throughout the source code of a program. The similar characteristics of aspects and clones lend to the extension of the Visualiser to accommodate the visualization of clones.

Our 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.

2. TOOL INTEGRATION & EXTENSION
The two main tools that are integrated and extended are briefly described in this section. CloneDR serves as an external pre-existing command-line clone detection tool that is called by the plugin and executes independently. The results of CloneDR are retrieved and parsed by our plugin and reported to the user. The visualization feature is implemented through an Eclipse plugin extension point provided by the AJDT Visualiser plugin.

2.1 CloneDR
CloneDR is a clone detection tool from Semantic Designs that is based on research described by Baxter et al. [1]. This tool examines the abstract syntax tree representation of a program where subtrees are matched according to a calculated similarity value to detect code clones in the program. It was one of tools evaluated in the *First International Workshop on the Detection of Software Clones*[2]. It is able to evaluate large software applications and is applicable to multiple languages. The version used in this paper is the Java version that is freely available[3].

2.2 AJDT Visualiser
The AJDT Visualiser is an Eclipse plugin that is part of the AspectJ Development Tools project. The developers of the Visualiser recently opened the plugin for adaptation by providing several extension points to allow other types of information to utilize its visualization features. Some examples of other tools that use the Visualiser include applications toward Google search results and CVS file histories[4].

[1] http://www.eclipse.org/ajdt

[2] http://www.bauhaus-stuttgart.de/clones
[3] http://www.semdesigns.com/Products/Clone/register-download.html
[4] http://www.eclipse.org/ajdt/visualiser

3. PLUGIN DETAILS

In this project, our plugin plays the role of an interface between CloneDR and the user. It performs the duties of setting up the configuration file, executing CloneDR as an external task, and displaying the clone detection results to the user. A diagram of the connections and processes related to this plugin is shown in Figure 1.

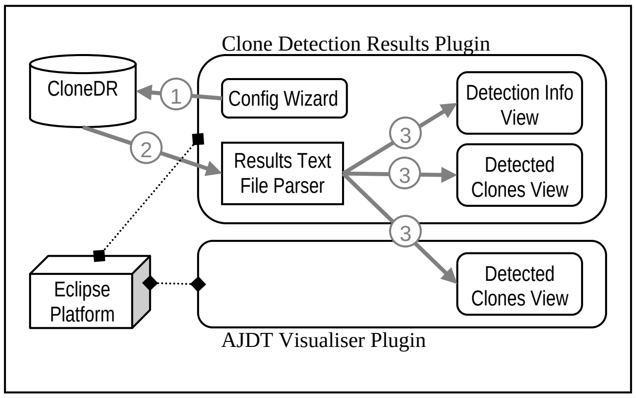


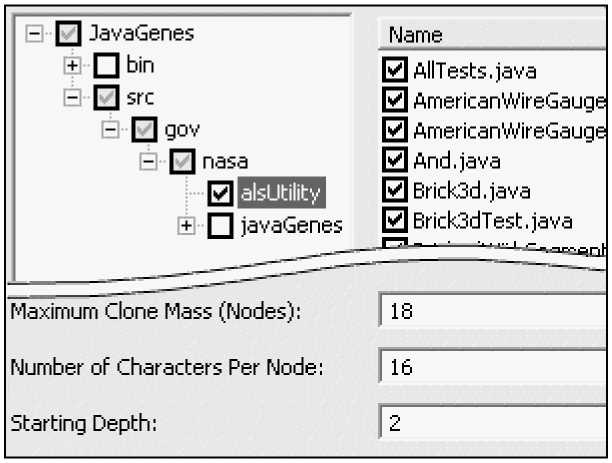
Figure 1. Plugin architecture and process

When the process is started (i.e., when the user clicks the clone detection button on the toolbar), the plugin will display a configuration wizard that assists the user to determine the type of configuration for the clone detection procedure. Upon completion of this wizard, the plugin will call CloneDR and give the location of the file containing the configuration settings (step 1). CloneDR will then execute and produce a text file containing its clone detection results. These results are parsed by the plugin (step 2) and sent to three Eclipse views that will display the results of the clone detection procedure (step 3). One of these views is an extension of the Visualiser that produces a graphical representation of the results of the clone detection. A more detailed description of the process is given in the following sections.

3.1 Configuration Setup

A file containing the configuration settings must be provided for every execution of CloneDR. The user is allowed to set five configuration settings. The values entered by the user determine the attributes that determine whether two sections of code are clones. Examples of these attributes include the similarity of code sections, the maximum number of parameters contained in each clone, and the starting depth of the subtree to be evaluated. In addition to the five configuration settings, the configuration file will also contain the list of all source files that will be searched for clones. Each absolute file location is placed as a separate line after the five configuration settings.

Our plugin contains classes that extend the Eclipse `INewWizard` and `WizardPage` classes to produce two wizard pages that assist the user with the configuration setup. The first wizard page contains a checkbox listing of files and directories of the Eclipse project that is currently selected and textboxes for the configuration settings. The directories are displayed on the left side using a default plugin class based on `ContainerCheckedTreeViewer`. The files for the directory that are selected on the left are displayed to the right (using `CheckboxTableViewer`). The detection of clones usually includes all files for a project or program, but this setup allows users to select specific files. The second wizard page assists the user to determine where to save the configuration file containing the settings and selected files. Figure 2 shows part of the first wizard page (the middle part is removed to save space).



Maximum Clone Mass (Nodes): 18

Number of Characters Per Node: 16

Starting Depth: 2

Figure 2. First page wizard

3.2 Clone Detection Execution

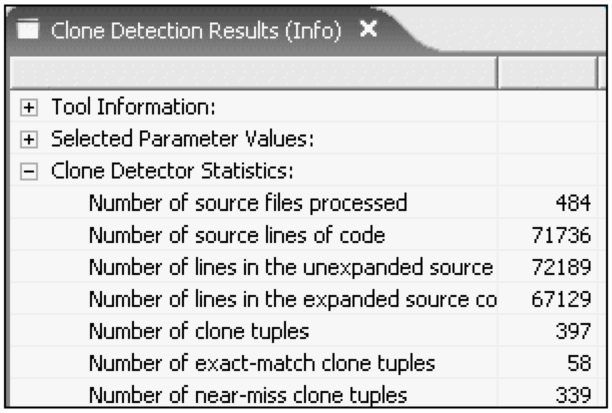
Our plugin invokes CloneDR as a separate command-line process. The arguments that are passed to the command include the location of the configuration file and the location of the file that will contain the results of the clone detection procedure. Typically, the name of the results file is the name of the configuration file with an added .log extension. The results file will be parsed by the plugin to extract the information about the detected clones.

3.3 Display of Clone Detection Results

After the clone detection procedure is completed, our plugin will display the results in three customized views:

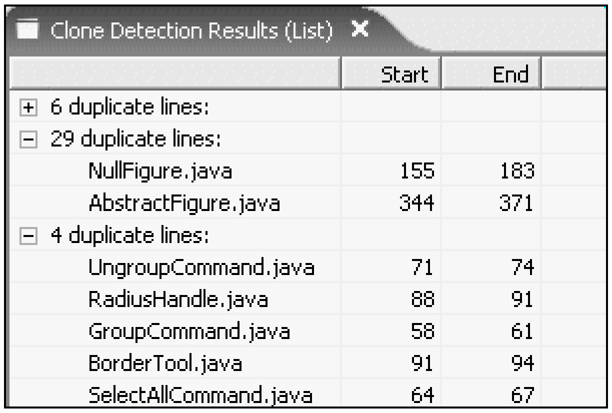
- General information view
- Detected clone list view
- Detected clone Visualiser view

The last two views essentially display the same information with the only difference in appearance - one is textual and the other is graphical. Both views update their states through listeners that are triggered when information about a clone is parsed and retrieved from the results file. A more detailed description of each view is given below.



⊞ Tool Information:	
⊞ Selected Parameter Values:	
⊟ Clone Detector Statistics:	
Number of source files processed	484
Number of source lines of code	71736
Number of lines in the unexpanded source	72189
Number of lines in the expanded source co	67129
Number of clone tuples	397
Number of exact-match clone tuples	58
Number of near-miss clone tuples	339

Figure 3. General information view



	Start	End
⊞ 6 duplicate lines:		
⊟ 29 duplicate lines:		
NullFigure.java	155	183
AbstractFigure.java	344	371
⊟ 4 duplicate lines:		
UngroupCommand.java	71	74
RadiusHandle.java	88	91
GroupCommand.java	58	61
BorderTool.java	91	94
SelectAllCommand.java	64	67

Figure 4. Detected clones list view

3.3.1 General information view

The information in the results file contains the location of clones in the source files, as well as general and statistical information. This information includes the values that were selected by the user for the various configuration settings and statistical information about the clone detection procedure itself. It is retrieved by parsing the results file and sending the values to a content provider class that implements the `ITreeContentProvider` interface. This interface was used to allow the information to be displayed in a tree structure. The tree has only two levels, but this allows information to be grouped separately and expanded and collapsed. Figure 3 contains a sample of this view.

3.3.2 Detected clones list view

Three pieces of information are required to determine where in the source code a clone can be found: name of the source file containing the clone, and the starting and ending line numbers of the section of code. Each clone must be associated with a group of similar clones. In the results file, the groups of clones are listed together, which allows extraction of clones by groups. This view is similar to the general information view in that it contains a `TreeViewer`, where each clone group is an element of the root, and the clones of each group are children of the clone group. This tree also consists of two levels. Double-clicking on one of the clones listed will display the clone (with its corresponding lines highlighted) in the editor view. A sample of this view is given in Figure 4. Displaying clones in a text list is similar to that used in the Eclipse plugins for Simian[5] and SimScan[6], which are also clone detection tools.

3.3.3 Detected clones Visualiser view

Extending the AJDT Visualiser plugin offers a graphical view of the clones that were detected in the source files. The Visualiser plugin provides extension points to allow the development of visualization of types of information other than aspects in program modules. Code clones and aspects are very similar to each other. Both usually occur multiple times and are scattered throughout the program. These characteristics provide a perfect fit for the use of the Visualiser for clone detection results. The "provider" extension point was used to populate the content of the custom Visualiser view.

The display view of the Visualiser consists of three parts: bars, stripes, and kinds. In the Visualiser view, a bar represents a source file and kinds are distinguished by different colors. Stripes can occur in more than one bar and each stripe can be associated with one or more kinds. In our tool, a stripe represents a clone and a kind represents a clone group. In this case, stripes are only associated with one kind because a clone is only associated with one clone group. Before the clones are retrieved from the results file, the bars for the view are generated by traversing through the Eclipse project and selecting resources that are source files. Once the bars have been generated, stripes can be added to them to display the clones in each source file. Similar to the detected clones list view, double-clicking on a stripe will display the clone (represented by that stripe) highlighted in the editor view. Figure 5 shows the Visualiser view of a clone detection result. In this view only source files that contain clones are displayed. The Visualiser allows for such filtering. Scrollbars at the bottom of the view (not seen in the figure) can be used if the view does not display all the bars containing clones.

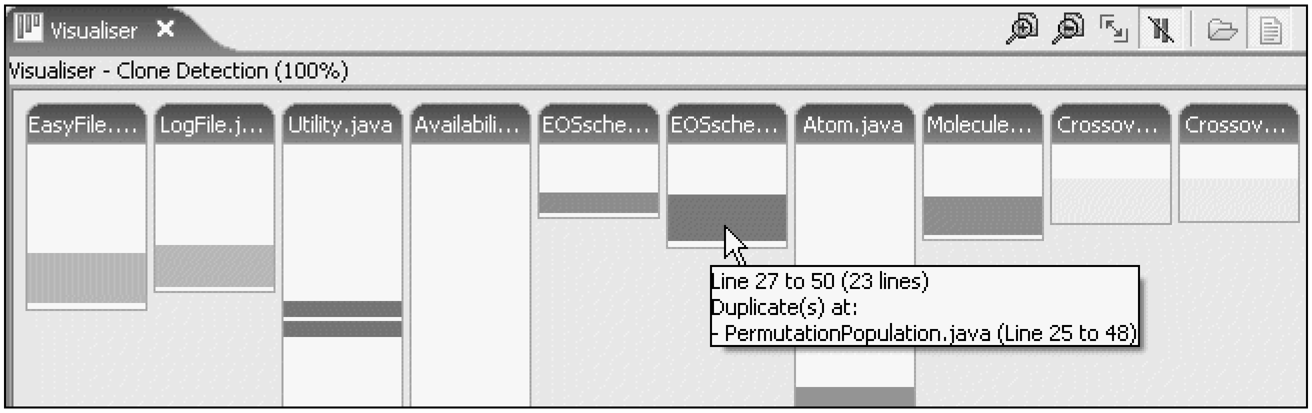
Figure 5 also displays an example of a popup text that provides more information about the clone when the stripe representing the clone is moused over. The information given includes the starting and ending line numbers of the clone in the source file and the names of other source files that also contain the stripe with their respective starting and ending line numbers.

3.3.4 Visualiser view enhancements

Taking advantage of the visualization offered by the Visualiser plugin through its extension points allows the clones to be displayed in a more visual manner. However, some limitations were found in the current state of the Visualiser. This is specifically true with the limitation of not being able to add customized actions on the context menu of the Visualiser view. One predefined action is given in the context menu, which emerges if the user right-clicks on a bar in the view. This action will display all bars that contain at least one of the stripes that is in the currently selected bar. An alternate filtering method could display the bars that contain the same type (kind) of stripe that is currently selected.

[5] http://www.integility.com/simian_ui

[6] http://www.blue-edge.bg/simscan/simscan_help_r1.htm



Line 27 to 50 (23 lines)
Duplicate(s) at:
- PermutationPopulation.java (Line 25 to 48)

Figure 5. Detected clones Visualiser view

When a user right-clicks on a stripe and selects this alternate functionality, only bars that also contain the selected stripe will be displayed. This is a useful feature in terms of clone detection results because only source files (or bars) that contain the same clone (or stripe) will be displayed. Clones that are scattered throughout the program can be separated out and viewed together. Figure 6 displays the two available options upon right-clicking an element in the view.

The context menu for the Visualiser view was not extendable because it was not registered. The source code of the Visualiser plugin was edited directly to allow the additional action in the context menu. Further enhancements to the Visualiser are discussed in Section 6 as future work.

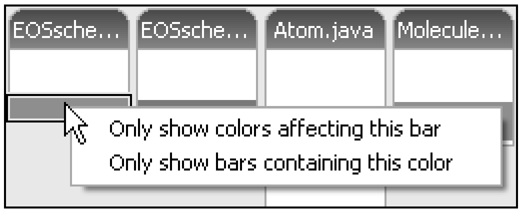

EOSsche...
EOSsche...
Atom.java
Molecule...
Only show colors affecting this bar
Only show bars containing this color

Figure 6. Context menu options

4. RESULTS REPRESENTATION

We applied our plugin on two open source programs, JavaGenes[7] and JHotDraw[8]. JavaGenes is an open source NASA program that has been made available to the public. JHotDraw is an open source GUI framework that is used frequently in software engineering tool evaluation. Both programs were written in Java.

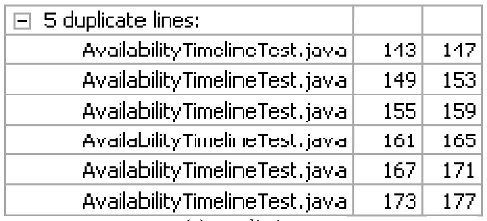
CloneDR was selected for this research because it is an established commercial clone detection tool that will be used for our future research work. However, the focus of this section is not about the quality of the clone detection results, but rather an investigation into the types of clone detection reporting methods. We implemented the three types of clone detection result representations: a text listing, an estimation of how a scatter plot would display the clones, and the new Visualiser extension.

[7] http://opensource.arc.nasa.gov/project.jsp?id=14

[8] http://www.jhotdraw.org/

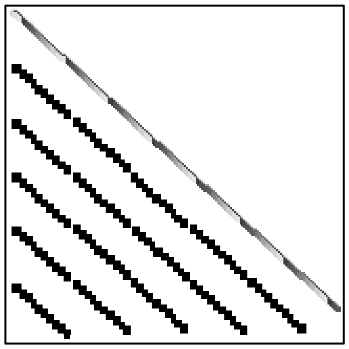
4.1 Clones in the same class

Both JavaGenes and JHotDraw contain an example where one group of clones is located in multiple locations in the same source file. In JHotDraw, `CTXWindowMenu.java` contains five sections of code that are the same clone. Similarly, the file named `AvailabilityTimeLineTest.java` in JavaGenes contains six sections of code that are clones in the same file. Figure 7 provides a sample of how the clones of the latter group are displayed in three different types of representations: (a) a text listing, (b) a scatter plot, and (c) a Visualiser view. The scatter plot consists of dots where each dot represents two statements that are duplicates of each other. When multiple dots generate a semblance of a diagonal line that is parallel to the main diagonal line, the section of code that is represented by the dots are clones. Compared to the diagonally separated lines, the Visualiser view provides a straightforward representation of the sections of code that are duplicated in a single source file. The clone group represented in Figure 7 corresponds to methods that contain identical statements with varying degrees of time values.

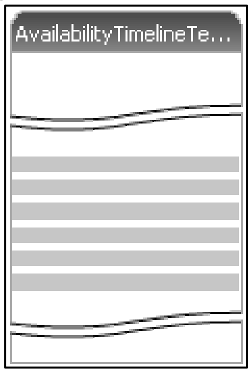


⊟ 5 duplicate lines:		
AvailabilityTimelineTest.java	143	147
AvailabilityTimelineTest.java	149	153
AvailabilityTimelineTest.java	155	159
AvailabilityTimelineTest.java	161	165
AvailabilityTimelineTest.java	167	171
AvailabilityTimelineTest.java	173	177

(a) text listing



(b) scatter plot



(c) Visualiser view

Figure 7. Three types of representations

4.2 Ubiquitous clones

As shown in Figure 8, several clones can be seen to be ubiquitous in that they are present throughout multiple source files in JHotDraw. The new feature in the Visualiser view can filter the bars representing the source files to show only those containing a particular clone type. This is a feature that scatter plots can not offer because non-connected sections that contain the same clone can not be grouped together. These ubiquitous clones in particular are short methods that perform a specific task, such as returning a new `Rectangle` object, drawing two ovals (including setting their colors), and setting the undo and redo flags for the drawing view.

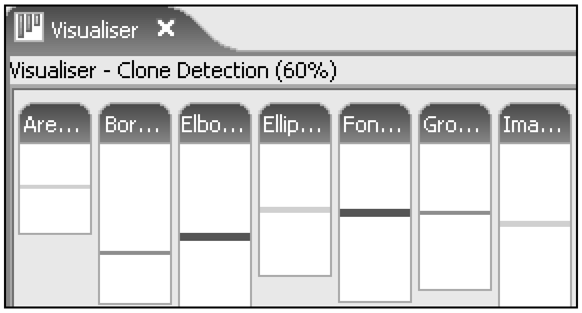


Figure 8. Ubiquitous clones

5. RELATED WORK

CCFinder [3] and Duploc [5] contain visualization of clone detection results using scatter plots. As seen in the previous section, this type of visualization has its limitations. As an alternative to scatter plots, CLICS [3] uses a visualization feature provided by a program called LSEdit that can display dependencies within the architecture of an application. In CLICS, the dependencies or relationships between entities represent the clones. Users can see how the entities in the architecture relate to each other through their cloned code.

Simian and SimScan are Eclipse plugins that contain the main components for the clone detection process. They allow users to setup the configuration for the search and once the search has completed, they display the results in views. However, the results are only represented in a textual form containing a list of clones.

6. CONCLUSION AND FUTURE WORK

Extending the AJDT Visualiser provides a new way of visualizing clone detection results in a manner that is observably different from the popular visualization using scatter plots. The clone visualization described in this paper compliments the standard text listing and scatter plot 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. Video demonstrations of our plugin can be found at: http://www.cis.uab.edu/tairasr/clones/visual

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. Additional features that would assist users to understand the clone detection results and enhance the clone detection process are planned as future work, 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.

REFERENCES

[1] Baxter, I., Yahin, A., Moura, L., Sant'Anna, M., and Bier, L. Clone Detection using Abstract Syntax Trees. In *Proceedings of the International Conference on Software Maintenance*, Bethesda, MD, November 1998, pp. 368-377.

[2] 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.

[3] Kapser, C. and Godfrey, M. Improved Tool Support for the Investigation of Duplication in Software. In *Proceedings of the 2005 International Conference on Software Maintenance*, Budapest, Hungary, September 2005, pp. 305-314.

[4] Kamiya, T., Kusumoto, S., and Inoue, K. CCFinder: A Multilinguistic Token-Based Code Clone Detection System for Large Scale Source Code, *IEEE Transactions on Software Engineering*, vol. 28, no. 2, July 2002, pp. 654-670.

[5] Rieger, M. and Ducasse, S. Visual Detection of Duplicated Code, in *Proceedings ECOOP Workshop on Experiences in Object-Oriented Re-Engineering*, LNCS 1543, Springer-Verlag, July 1998, pp. 75-76.