SoftGUESS: Visualization and Exploration of Code Clones in Context

Eytan Adar and Miryung Kim
University of Washington, Computer Science and Engineering
{eadar,miryung}@cs.washington.edu

Abstract

We introduce SoftGUESS, a code clone exploration system. SoftGUESS is built on the more general GUESS system which provides users with a mechanism to interactively explore graph structures both through direct manipulation as well as a domain-specific language. We demonstrate SoftGUESS through a number of mini-applications to analyze evolutionary code-clone behavior in software systems. The mini-applications of SoftGUESS represent a novel way of looking at code-clones in the context of many system features. It is our hope that SoftGUESS will form the basis for other analysis tools in the software-engineering domain.

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. To facilitate different levels of graphical analysis—anything from clones embedded in dependency diagrams to genealogies [10]—we implemented the SoftGUESS library. SoftGUESS is based on GUESS, the Graph Exploration System [1], a novel graph visualization and analysis program. GUESS distinguishes itself from other efforts by providing a domain-specific language for the manipulation, analysis, and visualization of graphs. Unlike tools which force overly-generalized or overly-specific graph representations, GUESS allows users in many fields—from social to computer to biological networks—to analyze their domain-specific graph representations.

SoftGUESS consists of a code library and a number of mini-applications that supports the analysis of code-clones in the context of system dependencies, authorship information, package structures, and other system features. SoftGUESS supports visualization of code clones in a single version or a program as well as views of changing clone over multiple version of the program. Code cloning behavior in software systems has been broadly studied (e.g. [2][6][8][9][10][11]) but primarily for the purposes of identifying targets for refactoring (e.g. [3][4][6]). One of the most recent of these studies [10] analyzed the code clone behavior of two medium size open source projects, *Carol* (carol.objectweb.org) and *dnsjava* (www.dnsjava.org). By analyzing multiple versions of each codebase, the authors of [10] studied how clones change over time.

In designing the components of SoftGUESS, we were motivated by previous visualization work on code-clones including graphs of genealogy [10], spectographs [15] , code evolution views [13], dot-plot views [5][8], Hasse diagrams [7], and polymetric views[14]. While all convey some information about the evolution of a codebase, these perspectives do not always allow us to understand clone evolution behavior in the context of other important system attributes, switch between contexts, or experiment with multiple contexts simultaneously. The ability to quickly move through these views, issue queries and visualize results—as static figures and dynamic animations—is beneficial for both researchers and end-users.

As an example, we imagine a scenario in which a set of clones diverge at some early time period and different developers take ownership of each copy. At some later time, one developer corrects a bug in the clone copy that is within their purview. A second developer, unaware of this fix, will not know whether to correct the bug. Using SoftGUESS, it is possible to quickly determine a) how often this phenomenon occurs, and b) to implement a check that traces back the history of the clone to the point of divergence and subsequently propagates a notification to authors responsible for different branches of the clone.

In another scenario a user may want to find locations in which the code-clone represents a large percentage of the content of the method, class or package that contains the clone snippet (i.e. segment). Methods that are largely or entirely copies of one another, and are called with the same parameters, may be eligible for refactoring. A simple query in SoftGUESS can find and visualize these spots in the code base.

Below, we begin with a brief introduction to GUESS and describe the major extensions built for SoftGUESS. Each extension provides a different view

of the clone genealogy dataset and provides a mechanism for asking questions on clone behavior.

2. Gython Basics

Gython [1] is an embedded Python/Jython based language and interactively controls GUESS. Nodes and edges in GUESS are first class objects which may hold any set of properties including strings, numbers, Booleans, and so on. Dynamic properties, such as degree or importance metrics (e.g. PageRank), are calculated on demand. Certain properties (e.g. color, width, label) have a meaning to the visualization subsystem (e.g. the command *node1.color = red* will set node1 to red and *(node1,node2).size = 10* will set both nodes to 10 pixels).

In addition, Gython pre-defines several methods, such as *getPredecessors*(), *getSucessors*(), *getOutEdges*(), etc, to access graph structures quickly. Edges are selected using special operators on individual nodes and node sets (e.g. *node1->node2*).

A more powerful feature of Gython is the built-in, SQL-like query language. For example, *snippet_size > 10* will return those nodes that are longer than 10 lines of code. GUESS supports most SQL operators (e.g. !=, <, >, like, etc.) as well as a few extras for range queries. This feature enables programmers to easily navigate and explore a particular set of code clones. For example, a programmer may want to focus on code clones with a substantial size in the networking subsystem. To visually highlight these clones, say for example by making them green, she could type: *((snippet_size > 10) & (package like '%network%')).color = green.*

Finally, Gython defines many shortcut methods to simplify certain analysis tasks. Frequently, users would like to change a visualization attribute based on some node or edge property (e.g. *colorize(file)* to color clone snippets from each file differently). This is similar to the polymetric views proposed in [14], but with a much richer set of visual properties which can be used to convey general system views. The commands *groupBy(…)* and *groupAndSortBy(…)* generate a set of sets corresponding to a parameter. For example, one could identify the shortest lived clone in each package by issuing the command: *for c in groupBy(package): (c.sortBy(lifespan)[0]).color = red*.

3. The Clone Genealogy Data Set

In constructing the clone dataset, it became apparent that there were an overwhelming number of possible visualizations. Clone snippets, methods, and classes can be represented as nodes; dependencies, inheritance and overloading relationships, containment, and cloning relationships can be represented as edges. Initially SoftGUESS concentrates on three specific visualizations which highlight a number of distinct relationship types (rather than combining them in an uninformative way): (1) a simple visualization of *clone evolution*, (2) clone evolution in conjunction with *containment relationships*, and (3) clone evolution in conjunction with *structural dependencies*. Each visualization shows clones in the context of source code (when a user clicks on a node), or a side-by-side view of clone changes when an edge is clicked.

For the purposes of this paper we make use of one particular clone genealogy dataset, specifically, the data generated by [8] for the *Carol* system. This data includes 37 versions over the course of 26 months. For each release, the complete dependency diagram was determined for each compilable version (using DependencyFinder, http://depfind.sourceforge.net/). Though more detailed views are possible, in order to manage the size of the graph, only incoming and

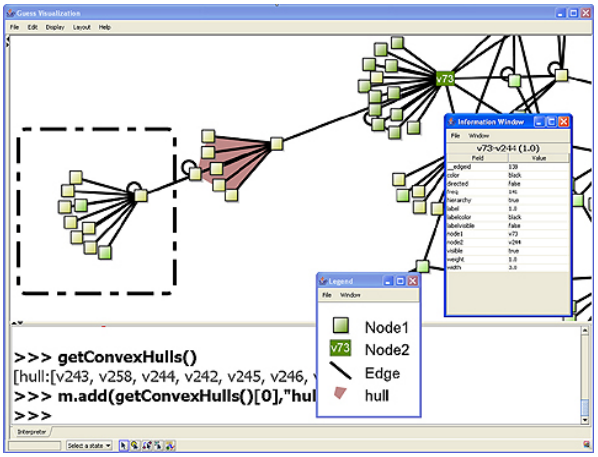

Figure 1: A screenshot of the GUESS system.

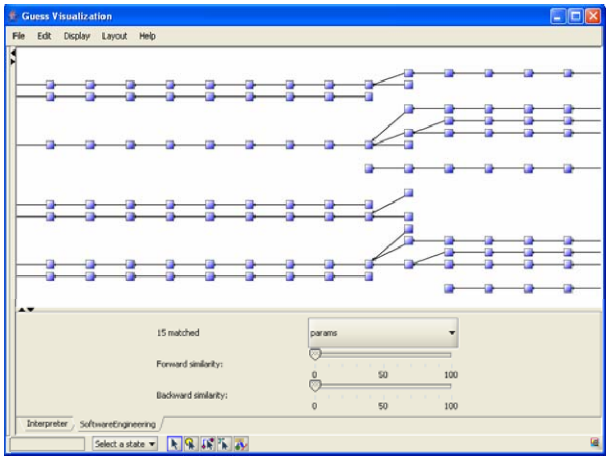

Figure 2: The Genealogy Browser

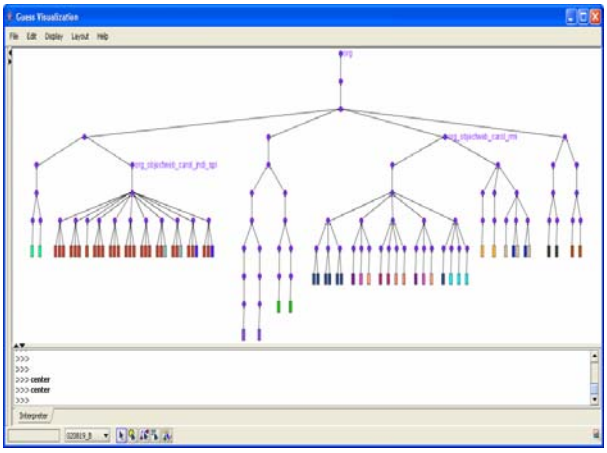

Figure 3: Encapsulation Browser

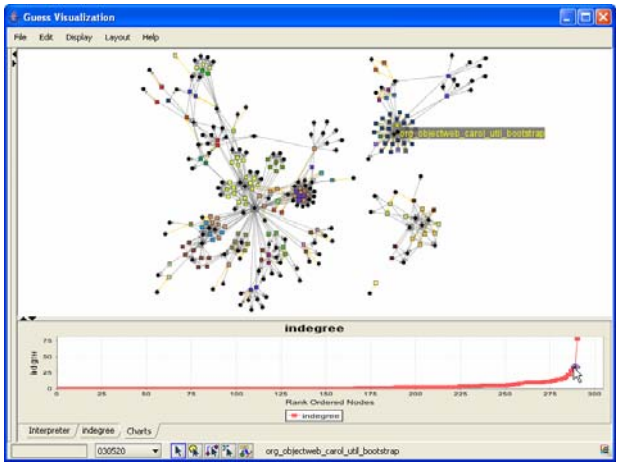

Figure 4: Dependency Browser

outgoing edges from the method, class, and package which contain or invoke clones were recorded.

Author information for each version was determined through CVS logs. Each snippet was also annotated with its size, the size of the method and containing class, and the number and type of parameters in the container method. Depending on the analysis mode, graphs for the *Carol* system contained 8000-8500 nodes (i.e. clone snippets) and up to 35,000 edges.

4. Genealogy Browser

The Genealogy Browser, as shown in Figure 2, is the simplest representation of clone genealogy data. Nodes, representing code clones, are positioned from left to right by version and clones from the same group are placed next to each vertically. This layout is made easy by the various clustering and graph processing methods built into GUESS

Since most code clones do not actually change between versions, the browser allows users to issue commands like *notEqualFilter(param)* which will find those edges where the end points change in parameter. For example, we may be able to spot potential trouble spots where clone ownership diverges by using *notEqualFilter(author)*. Other parameters include the *package*, encapsulating *method*, the method type (i.e. *public*, *private*, *protected*, or *static*), and the number and type of parameters (*params* and *param_types*). To support these certain exploration tasks we created a tool bar that allows the selection of the parameter. In all, the toolbar, libraries, and layout algorithm are implemented in under 250 lines of Gython code.

5. Encapsulation Browser

The Encapsulation Browser, which is similar to the system complexity view of [13][14], visualizes a tree representing hierarchical containment of clone snippets from the snippet itself (the leaves), through method, class, and package definitions. The resulting graph (8717 nodes and 16422 edges in the case of *Carol*) can be used to answer questions about the movement of clone snippets relative to each other in this hierarchy. Each code release version is set as a graph "state" and animated through GUESS' dynamic visualization feature. States are essentially a column in the Genealogy Browser, connected by the encapsulating Package/Class/Method (PCM) hierarchy to form a tree.

The layout algorithm, again simplified by GUESS features, produces trees such as the one in Figure 3. In this image the user has issued a *colorize(clone_id)* command, coloring each clone group differently. One can readily see the distribution of clones in the containment hierarchy. Furthermore, because GUESS allows for smoothing morphing between graph states, we are able to create an animated view (see: http://graphexploration.cond,org/softguess/) that illustrates the movement and spread of code clones over time. In this view, each clone appears at the location of its predecessor in the genealogy and moves to its location in the package hierarchy. A user is easily able to identify the clone that has been copied to a very distant package or class.

One task which we may wish to perform is finding packages where snippets with high average snippet to method length ratios. We can create a new parameter (*ms_ratio*) dynamically on nodes as in the following:

```
for z in (version == 0):
  leaves = findLeaves(z)
  sum = 0
  for k in leaves: sum = sum + k.ms_ratio
  z.ms_ratio = sum / len(leaves)
```

and visualize the results by resizing each node from low to high based on the percent using the command: *resizeLinear(ms_ratio,10,20).*

6. Dependency Browser

The final SoftGUESS mini-application is the Dependency Browser which represents the genealogy graph augmented with the dependency edges. With only classes in the *Carol* system represented, the graph has 8475 nodes (i.e. 172 PCM nodes) and 35746 edges of which the majority (25064) are incoming edges from external PCM nodes to the clone snippets, and only 2956 are outgoing edges from clone snippets to other *Carol* PCMs.

Visualizing all versions simultaneously is not particularly informative. Instead, a dependency graph is generated for each version by only rendering edges, PCMs and snippets present in that version (rendered as circles and squares respectively) through a force-directed layout technique. Incoming and outgoing edges are colored differently, allowing users to quickly get a sense of the distribution. Each clone snippet can be colored or sized based on different properties (e.g. the genealogy, the in-degree, etc.).

Using this graph a user can ask targeted questions such as: "how many different clone genealogies are depended on by a particular class?" or general ones such as, "which objects depend on, or are depended by, clones the most?" The answer for *Carol* is the TraceCarol object with 2985 incoming edges and RemoteShell object with 872 outgoing edges.

GUESS also provides basic charting methods. With the command *plotDistrib(indegree)*, a user can, for example, identify clone snippets that are lightly embedded in the dependency graph and may be easy to refactor. The output of this command is illustrated in Figure 4 with the rank-ordered in-degrees. The example output is illustrated in Figure 4. Mousing over the nodes in the top visualization will highlight their location in the plot. Similarly, mousing over the plot will highlight matching nodes in the graph.

6. Conclusions

The SoftGUESS library and mini-applications represent a mechanism by which code-clones can be visually and programmatically analyzed. The different forms of analysis made possible by understanding clones in the context of many other relationships are made simpler by the power of the GUESS system. A domain-specific language with which users can investigate clones in single version snapshots, as well as their changes over multiple versions, represents a powerful way to analyze graphs. It is our hope that the mini-applications and libraries created for SoftGUESS will form the basis for other software-engineering visualization and analysis tasks. In our continuing work, we are also interested in performing more evaluations and testing of the visualizations to validate their usefulness.

GUESS and SoftGUESS, which are implemented in a combination of Java and Gython, are freely available at http://www.graphexploration.org.

7. Acknowledgements

We would like to thank David Notkin for his advice in the implementation of SoftGUESS. Eytan Adar is funded by an ARCS and NSF Fellowship.

8. Bibliography

[1] Adar, E., "GUESS: a language and interface for graph exploration," CHI 2006, Montreal, Canada, Apr. 22-27, pp. 791-800, 2006.

[2] Baker, B. "A program for identifying duplicated code," *Computing Science and Statistics*, 24:49-57, 1992.

[3] Balazinska, M., E. Merlo, M. Dagenais, B. Lague, and K. Kontogiannis, "Measuring clone based reengineering opportunities." METRICS '99, Boca Raton, FL, Nov. 4-6, pp. 292-303, 1999.

[4] Burd, E., and J. Bailey, "Evaluating clone detection tools for use during preventative maintenance." SCAM'02, Montreal, Canada, Oct. 1, pp. 36-43, 2002.

[5] Ducasse, S., M. Rieger, and G. Golomingi, "Tools Support for Refactoring Duplicated OO Code," ECOOP '99, Lisbon, Portugal, 1999.

[6] Higo, Y., T. Kamiya, S. Kusumoto, and K. Inoue, "Refactoring support based on code clone analysis," PROFES '04, Kyoto, Japan, Apr. 5-8, pp. 220-233, 2004.

[7] Johnson, J. H., "Visualizing Textual Redundancy in Legacy Source," CASCON '94, Toronto, Canada, Oct 1 – Nov. 3, pp. 9-18, 1994.

[8] Kamiya, T., S. Kusumoto, and K. Inoue. "CCFinder: A multilinguistic token-based code clone detection system for large scale source code," *IEEE TSE.*, 28(7):654-670, 2002.

[9] Kim, M. and David N., "Using a Clone Genealogy Extractor for Understanding and Supporting Evolution of Code Clones," MSR '05, St. Louis, MO, May 17th, 2005.

[10] Kim, M., V. Sazawal, D. Notkin, and G. C. Murphy, "An Empirical Study of Code Clone Genealogies," ESEC/FSE '05 Lisbon, Portugal, Sep. 7-9, 2005.

[11] Krinke, J., "Identifying similar code with program dependence graphs." WCRE '01, Stuttgart, Germany, Oct. 2-5, pp. 301-309, 2001.

[12] Li, Z., S. Lu, S. Myagmar, and Y. Zhou. "CP-Miner: A tool for finding copy-paste and related bugs in operating system code," OSDI '04, San Francisco, CA, Dec. 6-8, pp. 289-302, 2004.

[13] Nierstrasz, O., "The Story of Moose," ESEC/FSE '05 Lisbon, Portugal, September 7-9, 2005.

[14] Rieger, M., S. Ducasse, and M. Lanza, "Insights Into System-Wide Code Duplication," WCRE '04, Nov 8-12, Delft, The Netherlands, pp. 80-89, 2004.

[15] Wu, J., R.C. Holt, and A.E. Hassan, "Exploring software evolution using spectographs," WCRE '04, Nov 8-12, Delft, The Netherlands, pp. 100-109, 2004.