Cloning and Copying between GNOME Projects

Jens Krinke, Nicolas Gold, Yue Jia
King's College London,
Centre for Research on Evolution, Search and Testing (CREST)
{jens.krinke,nicolas.gold,yue.jia}@kcl.ac.uk

David Binkley
Loyola University Maryland,
Baltimore, MD, USA
binkley@cs.loyola.edu

***Abstract*—This paper presents an approach to automatically distinguish the copied clone from the original in a pair of clones. It matches the line-by-line version information of a clone to the pair's other clone. A case study on the GNOME Desktop Suite revealed a complex flow of reused code between the different subprojects. In particular, it showed that the majority of larger clones (with a minimal size of 28 lines or higher) exist between the subprojects and more than 60% of the clone pairs can be automatically separated into original and copy.**

I. Introduction

The duplication of code is a common practice to make software development faster, to enable "experimental" development without impacting the original code, or to enable independent evolution [1]. Since these practices involve both duplication and modification, they are collectively called *code cloning* and the duplicated code is called a *code clone*. A *clone group* consists of code clones that are clones of each other (sometimes this is also called a *clone class*). During the software development life cycle, code cloning is an easy and inexpensive (in both effort and money) way to reuse existing code. However, such practices can complicate software maintenance so it has been suggested that too much cloned code is a risk, albeit the practice itself is not generally harmful [2]. Because of these problems, many approaches to detecting cloned code have been developed [3]–[10]. While methods to identify clones automatically and efficiently are to some extent understood, it is still disputable whether the presence of clones is a risk. To better understand why and how code is cloned, recent empirical studies of cloned code have focused mainly on examining the evolution of clones, such as whether cloned code is more stable or changed consistently [11]–[17].

A lot of research has been done on finding and identifying software clones, but without additional information it is impossible to distinguish the original from the copy. Most of the above empirical studies use version control systems to extract limited information about the originals and their copied clones; for example, when a clone appears in some previous version. However, so far there has been only two approaches [18], [19] to distinguish originals from copies.

Most version control systems have a 'blame' command which shows author and version information for each line in a file. This information, which includes the version when the line was added or last modified, can be used as a line age: if all lines in one clone have older versions than the lines in the other clone of a clone pair, then the clone with the older lines is most likely the original and the other the copy. However, usually, it is not that simple because the original and the copy may have been modified in turn after the copy was created.

This paper makes the following contributions:

- It extends previous work [19] to automatically distinguish between copy and original by allowing the clones of a clone pair to be in different systems.
- A case study on the GNOME Desktop Suite subprojects shows that the majority of larger clones (with a minimal size of 28 lines or higher) exist between the subprojects and more than 60% of the clone pairs can be automatically separated automatically into original and copied clone.

The following section presents background on clones and clone detection, the retrieval of version information, and the approach to distinguishing copied clones from original clones. The case study on the GNOME Desktop Suite is then discussed in Section 3. Related work is discussed in Section 4 and the last section concludes.

II. Background

This section presents the framework in which code clones, groups of code clones, and changes to code clones are defined. This is followed by a description of how version information is retrieved from version control systems and how it is mapped onto the source code lines.

A. Code Clones

Code clones are usually described as source code ranges (or fragments) that are identical or very similar. They are grouped into *clone groups* (sometime called *clone classes*) which are sets of identical or very similar code clones. A code clone $c = (s, l, f)$ is the source code range starting at line s with the following l lines of code in file f, thus the last line of the code clone is line number $s+l-1$. A clone group $G = \{c_1, \ldots, c_n\}$ is a set of n code clones $c_1, \ldots, c_n$, where each of the code clones is a clone of the others. A group consisting of two clones is a *clone pair*. The clone pairs of a group are generated by pairing all clones of a group.

For the purpose of this study, the effects of *split* or *fragmented* code clones are ignored. Such clones would consist of multiple source code ranges in the same file. An example of such a code clone is a source code range that is copied and additional source code subsequently inserted into the copied code. The code clones do not have to be disjoint: it is possible

for two code clones $c_1 = (s_1, l_1, f)$ and $c_2 = (s_2, l_2, f)$ to share a common source range ($min(s_1 + l_1, s_2 + l_2) > max(s_1, s_2)$).

B. Version Information

Most current version control systems can track changes to a file line-by-line to show for each line the version when the line was last changed. *CVS* has an "annotate" command and *subversion* names the command "blame" because it shows the version and the author ('to be blamed'). These commands give crude information about the origins of the code based on when it was last changed and who made that change.

Usually, the blame command retrieves the version information for the current version or for a specific version for one file or a list of files. In the following, the existence of a function $\mathrm{V}(f, n)$, which retrieves the version (age) of source code line n from source file f of the current version of the program is assumed. This function can be used to retrieve the version of each source line in a clone $c = (s, l, f)$ present in the current version of the program.

C. Classification of Clones

The version information can be used to classify a clone pair c_1, c_2 with $c_1 = (s_1, l_1, f_1)$ and $c_2 = (s_2, l_2, f_2)$ into specific patterns. First, it is assumed that both clones have the same length ($l_1 = l_2$). How to classify pairs with different lengths will be addressed later. A clone pair is *copied* if the versions of all lines in c_1 are either larger or smaller than the corresponding lines' versions in c_2:

$$\forall_{i=0\ldots l_1-1} \mathrm{V}(f_1, s_1 + i) < \mathrm{V}(f_2, s_2 + i)$$

or

$$\forall_{i=0\ldots l_1-1} \mathrm{V}(f_1, s_1 + i) > \mathrm{V}(f_2, s_2 + i)$$

If the first condition holds, c_1 is likely to be the original and c_2 the copy. If the second condition holds, it is the other way round.

The above classification can only be applied to clones of equal length and does not allow for small differences of a few lines. In the following an extension is introduced that allows some tolerance. To do this, a limited number of source lines in the clones of a pair may be removed.

The clones of a clone pair c_1, c_2 with $c_1 = (s_1, l_1, f_1)$ and $c_2 = (s_2, l_2, f_2)$ are said to be *copied with a tolerance of* t if after removing t source lines the resulting pair can be classified as copied according to the above classification. The removal of the same line in c_1 and c_2 will count as one removal.

III. Case Study

To discover interesting patterns in the history of the GNOME desktop suite, the scheme presented above was used to classify clone pairs in the subprojects of the GNOME Desktop suite as provided by the MSR Challenge[1]. It is first evaluated based on how many clone pairs can automatically be classified as 'copied'. Second, the result of the automatic classification is used to study the flow of copied code between subprojects.

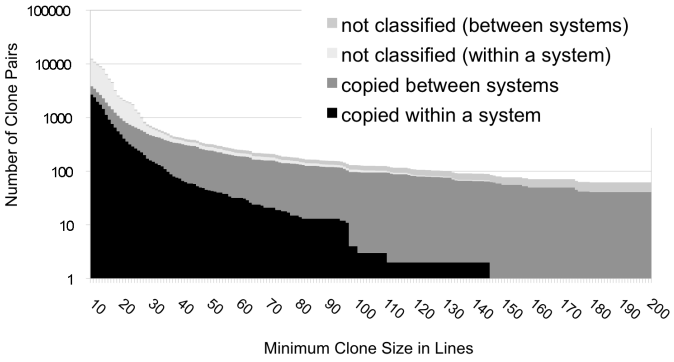

Fig. 1. Number of Clone Pairs

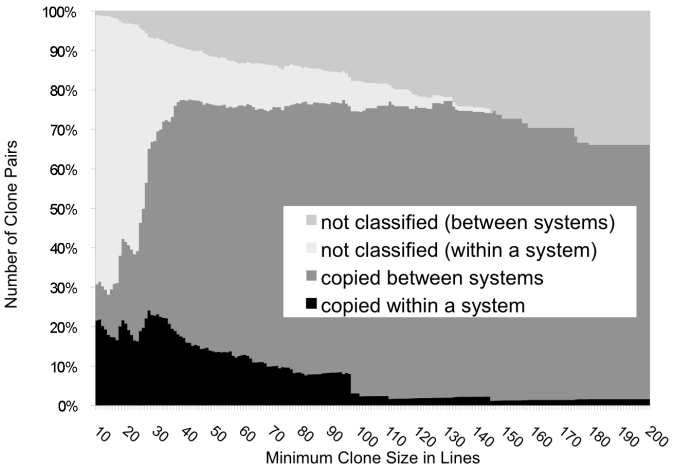

Fig. 2. Number of Clone Pairs (Percentages)

The approach uses Simian[2] (version 2.2.24) to identify the clones in a system and then applies the classification based on the version information available from the system's subversion repository. Simian was used to identify the clones in all C files of all subprojects. It has been used with the default settings except that the minimal size of a clone was set to 10 source code lines.

Simian identified 3096 clone groups, containing 8003 clones leading to 12512 clone pairs. Figure 1 shows the number of clone pairs for an increasing minimum clone size (x-axis, 10–200 source code lines) and how they are classified. The two lower landscapes with the darker colors show the number of clone pairs which could be classified as "copied" and separated into original and copy. The figure clearly shows that the number of clone pairs within a project drops much faster with increasing minimal clone size than the clone pairs between systems. The two upper landscapes with the lighter colors show the number of clone pairs that could not be classified (the light gray landscape is for clone pairs in between projects and the very light gray for clone pairs within a project).

[1] http://msr.uwaterloo.ca/msr2010/challenge/

[2] http://www.redhillconsulting.com.au/products/simian

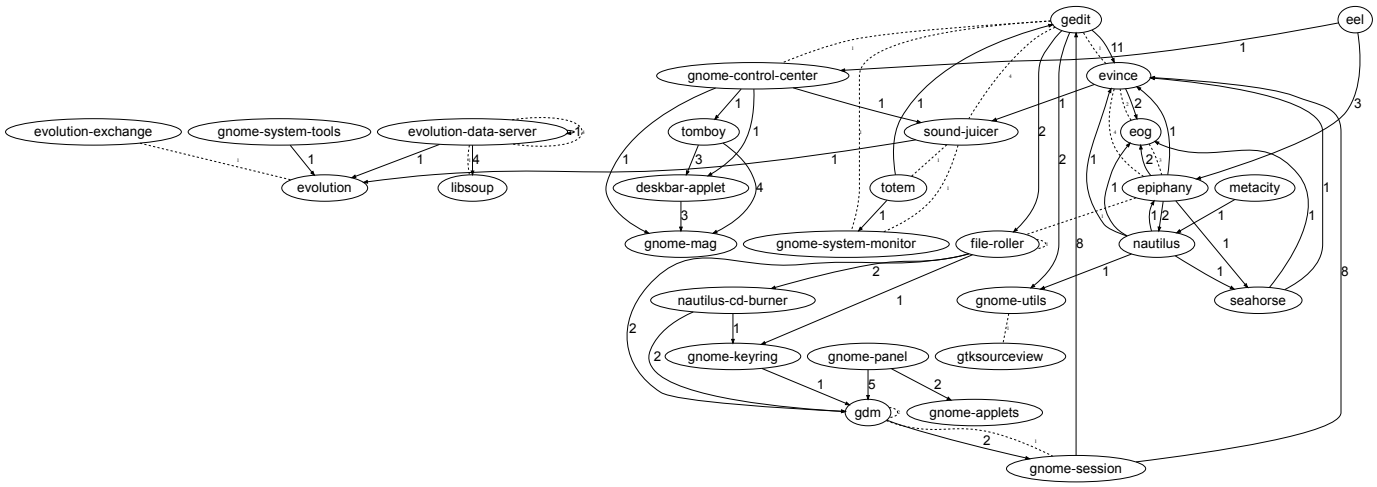

Fig. 3. Flow between Projects

The figure suggests that the majority of the smaller clones are clone pairs within a project and the majority of the larger clones are clone pairs between projects. Indeed, a look at the raw data reveals that this is true: The majority of clone pairs with minimal clone sizes of 28 and above are clone pairs between projects (383 pairs within a project vs. 397 pairs between project for a minimal clone size of 28 lines).

Because the number of clone pairs decreases asymptotically with an increased minimal clone size, it is reasonable to compare the data using percentages as shown in Figure 2. Again, it is clear that the number of clone pairs within a project decreases fast with increased minimal clone size. Indeed, there is only one single clone larger than 145 lines that appears within a single project. What is more important, is that the percentage of clone pairs between projects where it is possible to distinguish original and copy, is increasing fast with increased minimal clone size (although it is decreasing slowly for even larger minimal clone sizes). For all minimal clone sizes larger than 27 lines, at least 60% of the clone pairs can be separated into original and copied clone. A look at the raw data reveals that there exist 116 clone pairs between projects with a minimal size of 100 lines and only 14 clone pairs of that size within a project. For only 23 of the 116 clone pairs it was not possible to identify copy and original. This clearly shows that there are a lot of large clones in the GNOME Desktop Suite and the approach can automatically decide if a clone is a copy of another clone.

The large number of clone pairs where such an automatic classification is possible suggests that there is a lot of code reuse between the different projects in the GNOME Desktop Suite. Figure 3 shows the flow between the different projects. Solid directed edges show that code has been copied from one project to another and dashed edges show that there is shared code between projects where it was not possible to distinguish between copy and original. The edge labels show the number of code clones that are copied or shared. For example, eight clones have been copied from *gnome-session* to *gedit*. It is interesting to see that cycles in the graph are possible: For example, two clones are copied from *gedit* to *file-roller*, two clones are copied from *file-roller* to *gdm*, and two clones are copied from *gdm* to *gnome-session*.

Overall, the project *gedit* has the highest number of cloning relations with other projects: Eleven clones have been copied to *evince*, two to *gnome-utils* and *file-roller*, one from *totem*, and eight from *gnome-session*. Also, there are clones shared with *gnome-control-center*, *gnome-system-monitor*, *sound-juicer*, and *evince* where it was not possible to distinguish between copy and original. The graph also shows clusters of copying between projects: The projects *gnome-control-center*, *tomboy*, *deskbar-applet*, and *gnome-mag* build a cluster; projects *evolution-exchange*, *gnome-system-tools*, *evolution-data-server*, *libsoup*, and *evolution* build a cluster; projects *evince*, *eog*, *epiphany*, *nautilus*, and *seahorse* build a cluster.

The visualization as a graph clearly show the flow of code between projects in the GNOME Desktop Suite. It helps to understand the code reuse relationships between the different projects.

IV. Related Work

German et al. [18] used version information to identify the version where a clone has been introduced. For a clone pair between two systems it was then possible to identify the system where the cloned code appeared first. Together with license mining and classification they analyzed how copied code flows between Linux, FreeBSD and OpenBSD. Their approach used clone detection between the versions to track a clone as a clone pair between the current and previous versions. In contrast, the presented approach only uses a single invocation of a clone detector and utilizes the fine-grained line-by-line version information available in version repositories.

There are some studies on cloning across systems which are discussed below. None of them distinguished copies from originals. Al-Ekram et al. [20] studied the source code cloning across 17 systems (nine text editors and eight window

managers). They found very little cloning between systems and most of the cloning was accidental clones rather than real reuse of code. Unlike the above study, their study did not distinguished between large and small clones. Antoniol et al. [21] studied the extent and the evolution of code duplications in the nineteen releases of the Linux kernel. They found that the Linux system does not contain a relevant fraction of code duplication. Furthermore, code duplication tends to remain stable across releases, thus suggesting a fairly stable structure, evolving smoothly without any evidence of degradation. Kamiya et al. [10] studied the cloning between Linux, FreeBSD and NetBSD. They found a large number of clones between FeeBSD and NetBSD, but only a small number between Linux and the other two.

There are a lot of studies on the evolution of clones within projects. For example, Kim et al. [11] investigated the evolution of code clones and defined several evolution patterns to classify all possible changes during the clone evolution. Aversano et al. [12] did a similar empirical study with a slightly refined framework. Similar to Kim et al., they analyzed so called co-changes that are changes committed by the same author, with the same notes, and within 200 seconds (into a CVS repository). A similar framework and experiment was also presented by Krinke [14] to study the evolution of code clones with respect to consistent and inconsistent changes. The changes were reconstructed from data stored in a version control system (subversion or CVS). He found that the number of consistent and inconsistent changes were similar. In a second study, Krinke [15] investigated whether cloned or non-cloned code is more stable with respect to the number of changes applied to cloned and non-cloned code. Again, he reconstructed the changes from version control systems.

With a similar setup where changes are extracted from version control systems, Göde [16] presented a model for clone evolution where he tracked the evolution of individual clones throughout the history of a program. Thummalapenta [17] used an automatic approach to classify the evolution of source code clone fragments and investigated to what extent clones are consistently changed or evolve independently. Clone fragments were also tracked individually and the evolution of clones were classified into patterns.

V. Conclusions

The presented approach is able to separate the majority of clone pairs that occur within a system or between systems into the original and the copied clone. For the GNOME Desktop Suite the approach revealed a complex flow of reused code between the different subprojects. In particular, it showed that the majority of larger clones (with a minimal size of 28 lines or higher) exist between the subprojects and more than 60% of the clone pairs can be automatically separated into original and copied clone.

Acknowledgement

This work is funded in part by Hewlett Packard and the FOSSology Project.

References

[1] J. Cordy, "Comprehending reality – practical barriers to industrial adoption of software maintenance automation," in *11th IEEE International Workshop on Program Comprehension*, 2003, pp. 196–205.

[2] C. Kapser and M. W. Godfrey, ""Cloning considered harmful" considered harmful," in *13th Working Conference on Reverse Engineering (WCRE)*, 2006, pp. 19–28.

[3] J. Mayrand, C. Leblanc, and E. Merlo, "Experiment on the automatic detection of function clones in a software system using metrics," in *International Conference on Software Maintenance (ICSM)*, 1996, pp. 244–254.

[4] I. D. Baxter, A. Yahin, L. Moura, M. Sant'Anna, and L. Bier, "Clone detection using abstract syntax trees," in *International Conference on Software Maintenance (ICSM)*, 1998, pp. 368–378.

[5] B. S. Baker, "On finding duplication and near-duplication in large software systems," in *Second Working Conference on Reverse Engineering*, 1995, pp. 86–95.

[6] S. Ducasse, M. Rieger, and S. Demeyer, "A language independent approach for detecting duplicated code," in *International Conference on Software Maintenance (ICSM)*, 1999, pp. 109–118.

[7] K. Kontogiannis, "Evaluation experiments on the detection of programming patterns using software metrics," in *Fourth Working Conference on Reverse Engineering*, 1997, pp. 44–54.

[8] R. Komondoor and S. Horwitz, "Using slicing to identify duplication in source code," in *Eigth International Static Analysis Symposium (SAS)*, ser. LNCS, vol. 2126, 2001.

[9] J. Krinke, "Identifying similar code with program dependence graphs," in *Proc. Eigth Working Conference on Reverse Engineering*, 2001, pp. 301–309. [Online]. Available: citeseer.nj.nec.com/krinke01identifying.html

[10] T. Kamiya, S. Kusumoto, and K. Inoue, "CCFinder: A multilinguistic token-based code clone detection system for large scale source code," *IEEE Transactions on Software Engineering*, vol. 28, no. 7, pp. 654–670, Jul. 2002.

[11] M. Kim, V. Sazawal, and D. Notkin, "An empirical study of code clone genealogies," in *Proceedings of the 10th European software engineering conference held jointly with 13th ACM SIGSOFT international symposium on Foundations of software engineering (ESEC/FSE)*, 2005, pp. 187–196.

[12] L. Aversano, L. Cerulo, and M. D. Penta, "How clones are maintained: An empirical study," in *11th European Conference on Software Maintenance and Reengineering (CSMR)*, 2007.

[13] R. Geiger, B. Fluri, H. C. Gall, and M. Pinzger, "Relation of code clones and change couplings," in *9th International Conference of Funtamental Approaches to Software Engineering (FASE)*, ser. LNCS, no. 3922. Springer, Mar. 2006, pp. 411–425.

[14] J. Krinke, "A study of consistent and inconsistent changes to code clones," in *14th Working Conference on Reverse Engineering (WCRE)*, Oct. 2007.

[15] ——, "Is cloned code more stable than non-cloned code?" in *Eighth IEEE International Working Conference on Source Code Analysis and Manipulation*. IEEE Computer Society, September 2008, pp. 57–66.

[16] N. Göde, "Evolution of type-1 clones," in *Ninth IEEE International Working Conference on Source Code Analysis and Manipulation*. IEEE Computer Society, 2009, pp. 77–86.

[17] S. Thummalapenta, L. Cerulo, L. Aversano, and M. Di Penta, "An empirical study on the maintenance of source code clones," *Empirical Software Engineering*, March 2009.

[18] D. M. German, M. Di Penta, Y.-G. Gueheneuc, and G. Antoniol, "Code siblings: Technical and legal implications of copying code between applications," in *6th IEEE International Working Conference on Mining Software Repositories*. IEEE Computer Society, May 2009, pp. 81–90.

[19] J. Krinke, N. Gold, Y. Jia, and D. Binkley, "Distinguishing copies from originals in software clones," in *International Workshop on Software Clones*, May 2010.

[20] R. Al-Ekram, C. Kapser, R. Holt, and M. Godfrey, "Cloning by accident: an empirical study of source code cloning across software systems," in *International Symposium on Empirical Software Engineering*, 2005.

[21] G. Antoniol, U. Villano, E. Merlo, and M. Di Penta, "Analyzing cloning evolution in the linux kernel," *Information and Software Technology*, vol. 44, no. 13, pp. 755–765, Oct. 2002.