Code Clone Authorship — A First Look

Jan Harder
University of Bremen, Germany
harder@informatik.uni-bremen.de

Abstract

Code clones are said to threaten the maintainability of software systems. Changes to one cloned code sequence likely require propagation to its copies. Proper change propagation may be more difficult when the clones are created and maintained by different authors. We present an approach to track the authors of code clones and report on a first case study. The results indicate that the number of contributing authors has an effect on the probability that clones are changed and the probability that these changes are carried out consistently.

1 Introduction

The effect of clones on maintainability has been analyzed by various studies in the recent past. On the one hand, their results indicate that clones may cause maintenance problems. On the other hand, they show that clones can not be regarded as a threat to maintenance in general. This raises the question under which conditions clones threaten maintainability. One factor that makes clones harder to maintain may be the number of developers who contribute to a clone. For instance, when a developer copies code that is maintained by another developer, it may be more likely that these clones do not change consistently in the future. We previously discussed that the management of clones may be more difficult if multiple authors are involved. [3].

2 Authorship

This section describes our approach to track the authors of source code clones. First, we will define our tracking for source code in general and then apply it to code clones.

Code Authorship. To determine the authors of a source code sequence, we use a token-based approach that is based on the metadata of the *subversion* source code management system. The general idea is to annotate all tokens in all file revisions with the user who added them or modified them last.

Let $r_0, \ldots, r_n \in R$ be the set of revisions we analyze and $F(r_i)$ the set of source code files in revision r_i. Our approach consists of two steps. First, we build the token stream $T(f, r_0)$ for each source code file $f \in F(r_0)$. For each token $t \in T(f, r_0)$ we define the author of the token as $author(t)$, which is the user who committed r_0 to the repository. Second, we iterate over all upcoming revisions r_i with $0 < i \leq n$. If a file f was changed, we use the *Longest Common Subsequence* algorithm on its old token stream $T(f, r_{i-1})$ and its new token stream $T(f, r_i)$. The result provides the tokens that have been deleted and added in f in r_i. If a token is replaced with another one, this will be reported as a deletion and an addition. Let $TA(f, r_i)$ be the set of tokens that have been added to f in r_i ($TA(f, r_i) \subseteq T(f, r_i)$). For all $t \in TA(f, r_i)$ we define $author(t)$ to be the user who committed r_i to the repository. For all other tokens that were not changed $u \in (T(f, r_i) - TA(f, r_i))$ we define $author(u)$ as the author the token was assigned to in r_{i-1}.

The first step that handles the first revision r_0 must be changed if r_0 is not the initial revision of the project and contains source code that already has a history. This is the case when r_0 is not chosen as the first revision in the repository or if the repository was initialized with existing code (i.e., in a migration from another SCM system). In this case all tokens in all $f \in F(r_0)$ are assigned the artificial user *unknown* and the authorship information must be built from the following revisions until a significant amount of tokens has been mapped to their authors. This, however, does not apply for this study where we deliberately chose r_0 as the very first version of the system's history.

A sequence of source code S is a subsequence of the tokens from a file in one particular revision: $S \subseteq T(f, r_i)$. The tokens $t \in S$ may be assigned to different authors. Thus, a code sequence has a list of authors $a_1, ..., a_n$ of which each a_i with $1 \leq i \leq n$ contributed a subset of the tokens $S_{a_i} \subseteq S$. For this study we define the *main author* of a sequence as the author a_i who owns most of the tokens in S, that is the one with the highest $|S_{a_i}|$. If two or more authors contributed the same number of tokens, one author is chosen randomly as the main author.

One of the advantages of our token-based approach lies in its robustness against changes to the code layout. Line-based differencing techniques would report changes when a code sequence was only reformatted. In this case code formatting, which is often applied automatically, would change the authorship.

Clone Authorship. Source code clones are code sequences that are copied in different locations. We call

these sequences *clone fragments*. All clone fragments that are identical to each other from a *clone class*. Using our definitions for code authorship we can also define the authors of clone fragments. This is straightforward, because fragments are code sequences and, thus, have a list of authors as well as a main author. The main author of a clone class is defined as the author who is the main author of most fragments in the class. If two or more authors are main author of the same number of fragments, one of them is chosen randomly as the main author of the class. In the remainder of this paper we will refer to the main authors of fragments and classes simply as fragment author or class author.

3 Case Study

We conducted a first case study using our authorship tracking approach. The main goal was to evaluate the feasibility of our tracking approach and to gain first insights into clone authorship. Consequently, we chose a pragmatic setup using one of our own programs as subject system and detecting only exact clones.

Subject System. To start with the authorship analysis we analyzed the source code of our own clone detection tool named *clones* and the libraries it depends on. *Clones* is an active project that has been developed for more then 10 years. We analyzed all 1,516 svn revisions from october 2001 to march 2012. In this period *clones* steadily grew from 0 to 75 KSLOC of Ada code. A total number of 23 authors contributed to the code. That is, the system may contain clones that where created and maintained by different developers.

Clone Detection. We used our incremental clone detector [2] to detect clones in all *svn* revisions of the subject system. Only exact clones having fragments with a minimum length of 50 tokens were detected. During the incremental detection our tool tracks the token streams of all files and the changes applied to them. This allowed us to collect the authorship information in-process during clone detection.

4 Results

Across all revisions we detected 137,391 clone classes. That is, 91 clone classes per revision on average. We first analyzed how many clone classes have fragments with different authors. Among all clone classes we detected in all revisions, 33.7% had fragments with at least two different authors, whereas 66.3% of all clone classes have fragments from only one author. That is, most clones seem to be created by the author of the original code. Nevertheless, a significant amount of clone classes are created and maintained by different authors. The number of clone fragments in a clone class does not seem to be correlated with the number of fragment authors. We found many larger clone classes with up to 11 fragments that had only a single fragment author, but also many small classes with only two fragments which have different authors.

We further analyzed how likely it is that clone classes with single or multiple authors are affected by changes. Although changes to clones are rare, as was previously found in another study on this system [1], the probability that a class with different fragment authors changes is twice as high as the probability that a clone class with only a single fragment author changes.

Besides the probability of change we were also interested in the change propagation across the clone fragments. That is the question whether all fragments of a clone class change consistently (exactly the same change is applied) or inconsistently. Our results show that when clone classes with a single fragment author change, this happens consistently in 50.9% of all cases and inconsistently in 49.1%—this matches previous findings on the evolution of exact clones in this system [1]. This is different for clone classes with multiple fragment authors. These change consistently in only 20.4% of the cases, whereas 79.6% of the changes to these clone classes are inconsistent.

5 Conclusion and Future Work

We presented a technique to track the authors of cloned code fragments and conducted a case study that sheds first light on the effect multiple authors have on clones. Most clone classes were authored by a single user. Nevertheless, one third of the clone classes we found had multiple fragment authors. Clone classes with multiple authors are more likely to change than classes with only a single author. When a clone class with multiple authors changes, the probability of inconsistent changes is higher than for clone classes with single authors.

These results indicate the the authorship of clones does have an effect on how they evolve. Our future research will extend and refine the clone authorship analysis and investigate the effect of multiple authors in more detail. It will be directed to the causes, which will require qualitative analysis of the changes to clones with multiple authors. Furthermore, we will analyze further and larger systems. Problems related to multiple clone authors may be more likely in larger systems with more contributors. Different kinds of clones, such as near-miss clones, may also lead to different results.

References

[1] N. Göde. Evolution of type-1 clones. In *Working Conference on Source Code Analysis and Manipulation*, 2009.

[2] N. Göde and R. Koschke. Incremental clone detection. In *European Conference on Software Maintenance and Reengineering*, 2009.

[3] J. Harder and N. Göde. Quo vadis, clone management? In *International Workshop on Software Clones*, 2010.