Advanced Analysis for Code Clone Removal*

Sandro Schulze, Martin Kuhlemann
University of Magdeburg, Germany
{sanschul, kuhlemann}@iti.cs.uni-magdeburg.de

1 Introduction

Code cloning, i.e., the process of replicating code fragments by copy-paste(-and-adaption), is common in software development [1]. Although it comes along with some short-term advantages (e.g., time-to-market or reusing functioning code), it has crucial drawbacks in the long run, e.g., increased maintenance costs or inconsistent changes [1]. As a consequence, it is important to be aware of such code clones in the system. To this end, *clone detection techniques* have been proposed and widely applied [1]. However, the detection of clones is not enough but further processing steps are necessary to overcome or at least minimize the mentioned drawbacks.

One approach which aims at a durable elimination of code clones is refactoring [2]. Several approaches exist (e.g., [3, 4]), that differ in the underlying clone detection technique and thus, in the information provided for the refactoring process. However, each of these approaches has one or more of the following problems:

- Only some refactorings are provided (mostly *Extract Method* and *Pull Up Method* [2]) and thus, a considerable fraction of clones is not covered.
- The information passed to the refactorings allows only the removal of coarse-grained clones (e.g., functions) which furthermore have to be of specific types [5].
- All approaches provide an automated, non-interactive refactoring process. This, in turn, can lead to problems in maintainability and understandability of the refactored code.
- Only object-oriented refactorings *(OOR)* are considered, although there may be better approaches for single problems than OOR.

In this paper we introduce an approach for code clone removal which will tackle these problems. One main characteristic of this approach is to take into account detailed code clone analysis and classification as well as how the analysis results are presented to the user in order to guide an interactive removal process.

*The work was founded in parts by the Metop Research Institute, Sandtorstrasse 23, 39106 Magdeburg

2 Code Clone Analysis to prepare Refactoring

Our approach for supporting code clone removal is two-staged. In the first stage, a detailed analysis of detected code clones is performed. In the second stage, we focus on how the results of stage 1 can be presented in order to guide an interactive refactoring/clone-removal process. An abstract overview over the workflow of our approach is depicted in Figure 1. In the following we explain these stages in detail.

The Analysis Stage. This stage encapsulates a detailed analysis process, divided into one preprocessing and multiple postprocessing steps. The input for this stage are the detection results of existing clone detection tools. Currently, we are using CCFinder [6] as clone detector, but in general, any other tool is possible as well. In the preprocessing, we merge code clones, that have been detected to be similar to each other, to *clone classes*. After that, we classify these clone classes regarding the *type* of the cloned artifacts (e.g., function, loops etc.). Finally, we investigate the clone classes which can be decomposed into smaller clone classes. In case of finding, we divide the affected clone class.

Subsequently, the preprocessing phase starts, consisting of two main aspects: further classification and similarity measures. First, an *Abstract Syntax Tree (AST)* is generated for all code clones. Then, the clone classes are classified by the occurence of their members in the class hierarchy. Possible categories for this classifcation are *all code fragements with same superclass* or *code fragments with different superclass* as well as further nuances between these opposites. A second classification takes the occurence of clones in the file system into account as described in [7]. The information gained through categorization can be used for assessing heterogeneity or homogeneity of clone classes. Within a clone class each clone pair is compared line-by-line (using additional information from semantical analysis). The similarity of each line is evaluated according to the terminology of Koschke [5], i.e. two lines can be classified as *identical, similar* or *pretty differing*. To analyze the result, we relate the clones of a clone pair line-by-line in a matrix manner. Thus, we obtain a matrix for each clone pair of a clone class which we use for further reasoning. As a

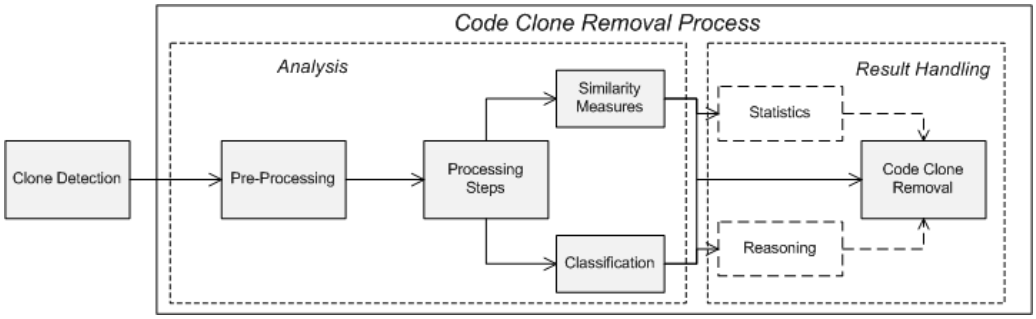

Figure 1: General Structure of the Code Clone Removal Process

next step, all (clone pair) matrizes of one clone class are merged in order to evaluate the similarity of the whole clone class. The benefit of this approach is, that we can determine the type of a clone class but beyond that, we can also determine *how much* of the clone class (and its clones) is of that type (according to the terminology in [5]). The similarity analysis is furthermore supplemented by the identification of referenced methods, fields etc. for each code clone. We use this detailed information, amongst others, to visualize differences between code clones in the second stage, e.g., by syntax highlighting.

The Result Handling Stage. This second stage presents the gathered information of the analysis stage to the user. Thereby, it is important to provide the user with different views on clones as well as with abstract information about every clone's origin. On this basis, we can now decide whether to remove them or not. For instance, we consider it useful to provide information about heterogeneity/homogeneity of the clone classes but also about their similarity. Further interaction options like browsing, searching, or filtering for certain types of clones (including all introduced classifications) are possible as well. To this end, this stage contains a *reasoning* and *statistics* part (cf. Figure 1).

Since we focus on code clone removal, all information finally form a removal proposal, consisting of one ore more refactorings. The following refactoring process will be semi-automatic, similar to the refactoring engine of Eclipse [8].

The result handling stage in general and the refactoring process in particular are highly related to tool support, that presents information to the user.

We currently develop such a tool that implements our vision. Until now, the preprocessing phase and small parts of the result handling stage are implemented. As a matter of future work, we want to provide rules, which allow to propose a multi-staged removal process (e.g., "first, apply refactoring X and then apply refactoring Y") to the user. Furthermore, we want to take aspect-oriented refactorings in account [9]. Technically, we intend to establish traceability for removed code clones in order to avoid decreased understandability. To this end, we present information to the user about previous code clone removals on the underlying code, e.g., by using annotations or a system-wide history.

References

[1] C. Roy and J. Cordy, "A Survey on Software Clone Detection Research," School of Computing, Queen's University at Kingston, Tech. Rep. 2007-451, 2007.

[2] M. Fowler, *Refactoring - Improving the Design of Existing Code.* Addison Wesley, 2000.

[3] M. Balazinska *et al.*, "Advanced clone-analysis to support object-oriented system refactoring," in *Proc. of the Working Conf. on Reverse Engineering*, 2000.

[4] M. Rieger, S. Ducasse, and G. Golomingi, "Tool Support for Refactoring Duplicated OO Code," in *Proc. of the Workshop on Object-Oriented Technology*, 1999.

[5] R. Koschke, "Survey of Research on Software Clones," in *Proceedings of Dagstuhl Seminar 06301: Duplication, Redundancy and Similarity in Software*, 2006, p. 24ff.

[6] T. Kamiya, S. Kusumoto, and K. Inoue, "CCFinder: A Multilinguistic Token-Based Code Clone Detection System for Large Scale Source Code," in *IEEE Trans. Soft. Eng.*, 2002.

[7] S. Schulze, M. Kuhlemann, and M. Rosenmüller, "Towards a Refactoring Guideline Using Code Clone Classification," in *2nd Workshop on Refactoring Tools, Companion of OOPSLA*, 2008.

[8] R. M. Fuhrer, M. Keller, and A. Kieżun, "Advanced Refactoring in the Eclipse JDT: Past, Present, and Future," in *Workshop on Refactoring Tools*, 2007.

[9] M. Monteiro and J. Fernandes, "Towards a Catalog of Aspect-Oriented Refactorings," in *Proc. of the Int'l Conf. on Aspect-Oriented Software Development*, 2005.