EasyMerge - Clone Code Refactor

Shengying Pan
University of Waterloo
s5pan@uwaterloo.ca

Haocheng Qin
University of Waterloo
h7qin@uwaterloo.ca

Yahui Chen
University of Waterloo
y556chen@uwaterloo.ca

ABSTRACT
Abstract goes here.

Categories and Subject Descriptors
D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

General Terms
Software Engineering, Recommendation System

Keywords
Software Engineering, Clone Detection, Code Refactoring, Recommendation System

1. INTRODUCTION
In software development, it's very common seeing developers reuse code fragments by copying and pasting with or without minor adaptation. Moreover, for large scale projects, developers are often too lazy to browse existing source files so that they may rewrite similar or even identical functions which were already in the code base. As a result, software systems often contain sections of code that are very similar, called code clones.

Previous research shows that a significant fraction (between 7% and 23%) of the code in a typical software system has been cloned [1] [3]. Many code clones in code bases are unnecessary duplications. Code duplication can be a significant drawback, leading to bad design, and increased probability of bug occurrence and propagation. As a result, it can significantly increase maintenance cost, and form a barrier for software evolution. By detecting, categorizing, and removing code clones, we can produce easier to understand, cleaner, and more reusable code.

Clone detection has been an avid research topic in the field of software engineering for decades. Fortunately, several automated techniques for detecting code clones have already been proposed. However, how to deal with detected clones, e.g. how to distinguish necessary clones from unnecessary ones and how to refactor code to remove unnecessary clones still remain a big problem in not only commercial but also academic domain. As a result, in this paper, we try to classify code clones and build a recommendation system called EasyMerge to help developers merge unnecessary clones on top of current state-of-the-art clone detection approach.

More specifically, we pick CloneDigger [2], an anti-unification duplicate code detection tool as our underlying clone detection approach, for its overall performance, coverage of multiple clone types, and availability. EasyMerge integrates CloneDigger as the pre-processing tool, analyze its output clone pairs, and recommend possible merges which can remove unnecessary clones without changing functionality of code base, creating reference conflicts, nor causing troubles to future code understanding or development.

2. BACKGROUND
3. CLONE CODE DETECTION
4. CLONE MERGING ALGORITHM
5. EXPERIMENTAL RESULTS
6. CONCLUSIONS
7. FUTURE WORK
8. ACKNOWLEDGMENTS
This section is optional; it is a location for you to acknowledge grants, funding, editing assistance and what have you. In the present case, for example, the authors would like to thank Gerald Murray of ACM for his help in codifying this *Author's Guide* and the **.cls** and **.tex** files that it describes.

9. REFERENCES
[1] B. Baker. On finding duplication and near-duplication in large software systems. *Proceedings of the 2nd Working Conference on Reverse Engineering*, pages 86–95, 1995.

[2] P. Bulychev and M. Minea. Duplicate code detection using anti-unification. *Spring Young Researchers Colloquium on Software Engineering, SYRCoSE 2008*, pages 4–7, 2008.

[3] C. K. Roy and J. R. Cordy. An empirical study of function clones in open source software systems. *Proceedings of the 15th Working Conference on Reverse Engineering*, pages 81–90, 2008.