Clone Detection and Refactoring

Robert Tairas
Department of Computer and Information Sciences
University of Alabama at Birmingham
Birmingham, AL 35294-1170
tairasr@cis.uab.edu

Abstract

The area of clone detection (i.e., searching for duplicate fragments of source code) has received wide interest recently as indicated by numerous efforts in clone detection tool development. Additionally, some work has been done in the area of selecting and refactoring the detected clones. However, the actual refactoring of clones is typically separated from the initial clone detection and subsequent selection of clones, because the task is delegated to a refactoring tool. This research abstract describes an investigation that will bridge the gap between clone detection and refactoring by researching and developing a comprehensive clone detection and refactoring process that begins with the detection of clones and ends with their refactoring.

Categories and Subject Descriptors D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement – *restructuring, reverse engineering, and reengineering*.

General Terms Design, Languages, Management.

Keywords clone detection, duplicate code, refactoring

1. Introduction

When a section of code is duplicated in more than one location among a collection of source files, that section of code and all of its duplicates are considered code *clones*. Clones are typically generated because a programmer copies a section of code and pastes it into other parts of the same program. Research has shown that a considerable percentage of code in large-scale commercial software (up to 5% in some observed case studies [2]) are clones. During evolution and adaptation of the source code, it is beneficial to determine if code clones occur in a program. If a section of code is known to be cloned, any updates to that section may necessitate the subsequent alteration of all clones of that section to perform the change consistently.

Depending on the size of the program, a manual process for finding duplicate code can become tedious and labor intensive. Various researchers have undertaken the development of automated clone detection tools by examining different levels of program representation. These investigations have used text-based, token-based, AST-based, program dependence graph-based, metrics-based, and information retrieval-based representations [3]. A bibliography of clone detection literature is provided at: http://www.cis.uab.edu/tairasr/clones/literature.

To eliminate the need to maintain clones in multiple locations and to improve the modularity of the program, refactoring activities on the clones can be performed. However, the aim is not just to reduce the lines of code of a program, but also to strike a balance between the efficiency and comprehensibility of the source code while improving change evolution during maintenance. Techniques to select clones for refactoring have been developed (discussed in the next section), but the extraction and refactoring of these selected clones remains a separate task that must be performed by the user. A mechanism that can perform the entire process would enhance the procedure of dealing with clones in source code.

OOPSLA'06 October 22–26, 2006, Portland, Oregon, USA.
ACM 1-59593-491-X/06/0010.

2. Clone Refactoring Research

Komondoor [6] looks for clones that can be refactored through procedural extraction and ARIES [5] uses customized metrics to determine clones that could be combined into a single new method (e.g., using the "Extract Method" and "Pull Up Method" refactorings). However, the user is left with the task of manually refactoring these clones and finding a refactoring tool to perform the task. Several refactoring browsers can automate this task, such as the refactoring support in Eclipse. The automation in a refactoring tool assumes that the section of code where the refactoring should be done has been selected by the user, which could be taken from the results of the tools described above. However, the process is manual and the user is given the responsibility to port the information from one tool (e.g., ARIES) to another (e.g., Eclipse). Balazinska et al. [1] introduce a tool that performs both the detection and refactoring of clones for object-oriented systems. The limitation of this tool is that only method level clones are considered.

Although the detection of clones has received considerable attention, only a few studies have focused on the refactoring of these clones. Moreover, these studies stop short of actually performing the refactoring. On the other side of the spectrum, tools for refactoring are widely available, but they require the user to determine where in the source code the refactoring should be done. A gap can be seen between these two areas.

3. Goal Statement

The purpose of this research is to bridge the gap between clone detection and refactoring by investigating a seamless process that begins with the detection of clones and ends with appropriate clones being refactored. An anticipated contribution of the research is a technique that can determine which detected clones should be considered for refactoring in addition to a seamless transition that is more automated between clone detection and refactoring. With respect to the latter, it may necessitate adjustments to the existing processes of one or both areas, or require inserting an adapter to connect the two areas together. Some questions to be investigated in this research include: Can the possible types of abstractions and refactoring opportunities on clones be identified? What is an effective way to determine clones

for refactoring to produce the defined abstractions? How feasible is it to combine the two areas of clone detection and refactoring into one process? How much more (or less) efficient is the approach that will be developed compared to other existing approaches? How can the voluminous results from clone detection and refactoring be visualized to a user?

4. Approach

As an introductory project in the area of clone detection, a first step of this research project was to develop a clone detection tool (as a plug-in for Microsoft Phoenix [7]) that searches for function-level clones by evaluating the abstract syntax tree representation of a program through suffix tree structures [9]. Additional information about this project is available at: http://www.cis.uab.edu/tairasr/clones. Future enhancements of this tool include adding support for more languages and detecting statement level and near exact clones.

To proceed into the process of evaluating clones for the potential of refactoring, the focus of the project will move to the development of an extension of an established commercial clone detection tool (e.g., Semantic Designs' CloneDR™ [4]). In collaboration with Semantic Designs, the future research for this dissertation will use CloneDR™ as the base detection tool, which is capable of detecting clones in multiple languages. The new research tasks will include methods for extracting and refactoring clones as program transformation tasks that are performed in response to user selection of the clone candidates. It is anticipated that the results of this research will lead to a seamless integration of the phases of clone detection, selection of clones, and extraction and refactoring in a more automated manner not available in existing tools that perform only the detection and selection phases.

Before any refactoring can be performed, the possible abstractions that can be generated from the clones will need to be determined. Section 2 lists work on procedural or method abstractions, but we believe other potential abstractions exist beyond "Extract Method." For example, the recent area of aspect mining [3] presents another example of emerging clones. After these abstractions are categorized, the next step will be to develop methods to select the clones that can be refactored to produce these abstractions. These methods will be used on the initial group of detected clones to find specific candidate clones for refactoring.

Refactoring of clones may require customized processes in addition to the current automated refactoring methods. DMS (Design Maintenance System), which is the underlying infrastructure used by CloneDR™, will be beneficial in the refactoring stage. As a program transformation and software reengineering toolkit, DMS will serve as the core transformation engine for the extraction and refactoring tasks.

5. Evaluation

Typical experimental validation in clone detection research is usually performed by manually checking the results of the tool execution, or by first manually searching for clones in the source code and then verifying that the clone detection tool identifies these clones. However, these methods only focus on the identification of clones in general. An additional step will be needed to evaluate the detection of clones that can be refactored. As artifacts for experimentation, the research will examine programs used in the evaluation of clone detection tools and also previously unused classes of programs, such as open-source programs written by NASA [8], to determine the degree to which clones can be discovered, with subsequent refactorings performed.

A research question that will be addressed concerns the ultimate question of whether the quality of the program has been enhanced by the refactoring and the removal of clones. Quality may pertain to maintainability, performance, or comprehensibility of code. These factors will be considered in the research evaluation in lieu of the research questions mentioned in Section 3.

6. Current Work

A current focus of this work is an investigation into visualization of identified clones. Currently, we are working on an Eclipse plug-in that serves as a graphical frontend for CloneDR™. In addition to displaying the results of the clone detection process in a customized Eclipse view, the results are also displayed graphically in an extension of the AspectJ Development Tool (AJDT) Visualiser tool. More information is available on the project's web site at: http://www.cis.uab.edu/tairasr/clones/visual. This work is not directly associated with the main purpose of our research, which is the selection of clones for refactoring and the refactoring activity itself. However, it is needed to help us prepare CloneDR™ for the subsequent refactoring activities.

References

[1] Balazinska, M., Merlo, E., Dagenais, M., Lagüe, B., and Kontogiannis, K. Advanced Clone-analysis to Support Object-oriented System Refactoring. In *Proceedings of the Working Conference on Reverse Engineering*, Brisbane, Australia, November 2000, pp. 98-107.

[2] Baxter, I., Yahin, A., Moura, L., Sant'Anna, M., and Bier, L. Clone Detection using Abstract Syntax Trees. In *Proceedings of the International Conference on Software Maintenance*, Bethesda, MD, November 1998, pp. 368-377.

[3] Bruntink, M., van Deursen, A., van Engelen, R., and Tourwé, T. On the Use of Clone Detection for Identifying Crosscutting Concern Code, *IEEE Transactions on Software Engineering*, vol. 31, no. 10, October 2005, pp. 804-818.

[4] CloneDR™, http://www.semdesigns.com/Products/Clone

[5] Higo, Y., Kamiya, T., Kusumoto, S., and Inoue, K. ARIES: Refactoring Support Environment Based on Code Clone Analysis. In *Proceedings of the 8th IASTED International Conference on Software Engineering and Applications*, Cambridge, MA, November 2004, pp. 222-229.

[6] Komondoor, R. V. *Automated Duplicated-Code Detection and Procedure Extraction*. Ph.D. Thesis, University of Wisconsin-Madison, Madison, WI, 2003.

[7] Microsoft Phoenix, http://research.microsoft.com/phoenix

[8] NASA Ames Research Center Open Source Software, http://opensource.arc.nasa.gov

[9] Tairas, R. and Gray, J. Phoenix-Based Clone Detection Using Suffix Trees. In *Proceedings of the 44th ACM-SE Conference*, Melbourne, FL, March 2006, pp. 679-684.