A Holistic Approach for Processing of Detected Code Clones

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

Abstract

Code cloning, i.e., the replication of code fragments in software systems, is a severe problem in industrial as well as in open source systems. In comparison to clone detection and analysis, which is a widely covered topic in research, only little work has been done regarding comprehension of clones or further processing strategies. Our current research aims at classifying code clones by adding semantic and syntactic informations in order to eliminate them durable, including a semi-automated refactoring process and source code enrichment which ensures maintainability and understandability of the overall system.

Code duplication is considered to be a common and harmful practice which evokes various problems, e.g., increased maintenance costs, bloated or dead code and others [1], [2], [3]. To detect such code clones, different techniques exist, such as string-based [2], metric-based [4], token-based [3] or tree/graph-based [5]. However, after the detection step, these approaches (or rather tools implementing them) stop without useful information about the code clones or subsequent clone processing (e.g., elimination). In some cases, visualizations or metrics are provided, reflecting the detected clones, but mostly, this information is blurred and thus, rather obfuscates the user.

Although some work has been done on comprehension and reengineering of duplicated code in the last years [6], [7], it is our position that there is still a need for a holistic approach, supporting the comprehension of code clones as well as their elimination. We initally started with a quite naive approach, classifying clones by location and type, along with some refactoring proposals for their removal [8]. In our current research, we focus on refining this approach. To this end, we use an *Abstract Syntax Tree (AST)* so that corresponding code clones (aka *clone classes*) can be analyzed in a fine-grained manner, e.g., by exploiting the inheritance hierarchy or comparing referenced methods and fields. This differs from existing approaches, where ASTs are only used for code clone detection (e.g., [5]) rather than for clone processing. With our approach, we not only classify code clones by their location/type but also by different degrees of similarity (obtained from the AST analysis).

In a subsequent step, the classification results are assigned to *elimination proposals*, using techniques from object-oriented refactoring (*OOR*) and aspect-oriented refactoring (*AOR*). The user then is guided through the refactoring process in a semi-automated manner, i.e., the decision if and what refactoring (in the case that there are alternatives) should be applied remains at the user, but the final process is automated as far as possible. This differs noticeable from the work of Balazinska et. al [6] in that we (1) not only consider function clones, (2) take aspect-oriented refactoring into account and (3) involve the user in the refactoring process which aims at a better comprehension of the resulting code. Additionally, we think about mechanisms which support the traceability of refactored clones like special kinds of comments or annotations. We think this is crucial to keep the maintainability and readability of the system, which may be decreased during the refactoring process [7]. Currently, we are working on tool support as an Eclipse Plug-In, which encompasses all our ideas, mentioned above.

References

[1] B. S. Baker, "On Finding Duplication and Near-Duplication in Large Software Systems," in *Proc. Working Conf. on Reverse Engineering (WCRE)*, 1995.

[2] S. Ducasse, M. Rieger, and S. Demeyer, "A Language Independent Approach for Detecting Duplicated Code," in *Proc. Int'l Conf. on Software Maintenance*, 1999.

[3] T. Kamiya, S. Kusumoto, and K. Inoue, "CCFinder: A Mulitlinguistic Token-based Code Clone Detection System for large scale Source Code," in *Trans. on Soft. Eng.*, 2002.

[4] J. Mayrand, C. Leblanc, and E. Merlo, "Experiment on the Automatic Detection of Function Clones in a Software System Using Metrics," in *Proc. of Int'l. Conf. on Software Maintenance*, 1996.

[5] I. D. Baxter *et al.*, "Clone Detection Using Abstract Syntax Trees," in *Proc. of Int'l. Conf. on Software Maintenance*, 1998.

[6] M. Balazinska *et al.*, "Advanced Clone Analysis to Support Object-Oriented System Refactoring," in *Proc. of Work. Conf. on Reverse Engineering*, 2000.

[7] C. Kapser and M. Godfrey, "Cloning onsidered harmful considered harmful: patterns of cloning in software," *Empirical Software Engineering*, 2008.

[8] S. Schulze, M. Kuhlemann, and M. Rosenmüller, "Towards a Refactoring Guideline Using Code Clone Classification," in *Proc. of Workshop on Refactoring Tools*, 2008.