A Taxonomy of Clones in Source Code: The Re–Engineers Most Wanted List

Cory Kapser and Michael W. Godfrey
Software Architecture Group (SWAG)
School of Computer Science, University of Waterloo
{cjkapser, migod}@uwaterloo.ca

Abstract

Code cloning — that is, the gratuitous duplication of source code within a software system — is an endemic problem in large, industrial systems [6, 5]. While there has been much research into techniques for clone detection and analysis, there has been relatively little empirical study on characterizing how, where, and why clones occur in industrial software systems. Our current research is to perform an in-depth analysis of code cloning in real software systems and to build a taxonomy of types of code duplication.

Code duplication, or code cloning, is generally believed to be common in software systems [6, 10, 12, 11, 8, 1, 5]. Various problems are associated with code duplication, including increased code size and increased maintenance costs. Other problems associated with code duplication can be found in [6, 12, 8, 1, 5]. While clone detection is an area of active research, and several tools exist to facilitate code clone detection, there has been relatively little empirical research on the types of clones that are found, or where they are found.

A code clone pair is a pair of source code segments that are structurally or syntactically similar. One of the segments is usually a copy of the other, perhaps with minor changes. Code cloning occurs when developers create two identical or similar code artifacts inside a software system. For example, developers may copy and paste code. Several methods exist for detecting code clones in software, such as simple string matching [6], using statistical fingerprints of code segments [7], function metrics matching [10, 12, 11], parameterized string matching [1, 8], and program graph comparison [5].

In our current research, we aim to profile the code cloning activity within software systems and generate a taxonomy of types of code duplication. In doing so, we hope to gain more insight into how and why developers duplicate code, in an effort to aid the development of code clone detection techniques and code clone elimination strategies. This taxonomy will consist of the most commonly occurring clones, and the clones with the strongest negative feedback. We plan to use large, industrial size software for our case studies, which will provide us with a better view of code cloning in the real world.

It is our position that generating a taxonomy of clones will provide several important contributions to clone detection research. This research will provide information about what kinds of clones are most prominent in software systems and how they are structured. This will give researchers more information on how to develop and tune their clone detection techniques. A taxonomy will provide an informative view of the types of code duplication that exists in software. It will also provide a more informative way of displaying code clones detected within a software system. The taxonomy can be used as a way of evaluating clone detection techniques and provide feedback about where and when to use them. This is important as it is still unclear which clone detection techniques are strongest.

In [9] we have begun our initial research into this taxonomy. There we have performed a case study on the Linux kernel file-system subsystem. We categorized different types of cloning activity using attributes such as location and scope after manual inspection of code clones found in the system. Each clone type provides a description of where the clone typically occurs, why it might arise, the problems associated with this type of clone, and possible solutions to the problem. We provide empirical analysis of these categories, and validation of our results using two different clone detection techniques. In future research we will try to incorporate more clone detection techniques, as this will ensure larger coverage of the code clones within a software system.

Other work tries to categorize clones for the purpose of software maintenance. In [2], Balazinska et al. create a schema for classifying various cloned methods based

on the differences between the two functions which are cloned. The results produced in [2] are used by Balazinska et al. in [4, 3] to produce software aided re engineering systems for code clone elimination. This differs from our work in that our classification scheme is based on locality as well as clone type, and copied functions are only one type in our case, although in [2] they break this down into 18 categories. One of our main research goals is to determine how much developers clone and from where. This question is not answered by the clone classification scheme in [2]. In addition, the work in [2] ignores code clones which are not function clones.

It is our position that this is an important step in understanding the phenomenon of code cloning and it will provide critical information to be used in enhancing and building clone detection software. Through case studies and further improving our taxonomy, we will gain valuable information such as what types of clones provide the largest refactoring gains, what types of clones are the most common, what clones are the most deadly, and how software requirements and design goals affect the practice of clone duplication. This information will help us evaluate current clone detection techniques, and current code clone display and elimination methods. It will also help guide us in uncovering problems in clone duplication research which need more work.

In the short term our research goals are to expand our case study to other subsystems in the Linux kernel, more specifically the driver subsystems, and to further expand and refine our taxonomy. Points of refinement are sub-classing clone types such as blocks within the same function. Our long term goals are to use the taxonomy to study the evolution of code clones in a software system. For example, we suspect that many code block clones that occur across subsystems begin as full function clones. We would like to test this hypothesis by comparing clone detection results from multiple versions of a software system. We also plan to study how the taxonomy can be used to improve the presentation of detected clones to software maintainers.

References

[1] B.S. Baker. A program for identifying duplicated code. In *Proceedings of Computing Science and Statistics: 24th Symp. Interface*, pages 49–57, 1992.

[2] M. Balazinska, E. Merlo, M. Dagenais, B. Lague, and K. Kontogiannis. Measuring clone based reengineering opportunities. In *Proceedings of the Sixth International Software Metrics Symposium*, pages 292–303, 1999.

[3] Magdalena Balazinska, Ettore Merlo, Michel Dagenais, Bruno Lague, and Kostas Kontogiannis. Partial redesign of java software systems based on clone analysis. In *The Proceedings of the 6th. Working Conference on Reverse Engineering*, pages 326–336, 1999.

[4] Magdalena Balazinska, Ettore Merlo, Michel Dagenais, Bruno Lague, and Kostas Kontogiannis. Advanced clone analysis to support object-oriented system refactoring. In *Proceedings of the 7th. Working Conference on Reverse Engineering*, pages 98–107, 2000.

[5] Ira D. Baxter, Andrew Yahin, Leonardo M. De Moura, Marcelo Sant'Anna, and Lorraine Bier. Clone detection using abstract syntax trees. In *ICSM*, pages 368–377, 1998.

[6] Stéphane Ducasse, Matthias Rieger, and Serge Demeyer. A language independent approach for detecting duplicated code. In Hongji Yang and Lee White, editors, *Proceedings ICSM'99 (International Conference on Software Maintenance)*, pages 109–118. IEEE, 1999.

[7] J. H. Johnson. Substring matching for clone detection and change tracking. In *Proceedings of the International Conference on Software Maintanence*, pages 120–126, 1994.

[8] Toshihiro Kamiya, Shinji Kusumoto, and Katsuro Inoue. Ccfinder: A multilinguistic token-based code clone detection system for large scale source code. In *Transactions on Software Engineering 8(7)*, pages 654–670. IEEE Computer Society Press, 2002.

[9] Cory Kapser and Michael W. Godfrey. Toward a taxonomy of clones in source code: A case study. In *Evolution of Large Scale Industrial Software Architectures*, 2003.

[10] K Kontogiannis. Evaluation experiments on the detection of programming patterns using software metrics. In *Proceedings of Working Conference on Reverse Engineering*, pages 44–55. IEEE Computer Society Press, 1997.

[11] K. Kontogiannis, R. De Mori, R. Bernstein, M. Galler, and E. Merlo. Pattern matching for clone and concept detection, 1996.

[12] J. Mayrand, C. Leblanc, and E. Merlo. Experiment on the automatic detection of function clones in a software system using metrics. In *Proceedings of the International Conference on Software Maintenance*, pages 244–253. IEEE Computer Society Press, 1996.