Electronic Communications of the EASST
Volume 63 (2014)

Proceedings of the Eighth International Workshop on Software Clones (IWSC 2014)

Investigating Intentional Clone Refactoring

— Position Paper —

Wei Wang and Michael W. Godfrey

6 pages

Guest Editors: Nils Göde, Yoshiki Higo, Rainer Koschke
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Investigating Intentional Clone Refactoring

Wei Wang and Michael W. Godfrey[1]

David R. Cheriton School of Computer Science
University of Waterloo, Canada
w65wang@uwaterloo.ca, migod@uwaterloo.ca

Abstract:

Software clone refactoring has been studied from many perspectives, including empirical research on clone refactoring history, IDE support for tracking clone change, and recommendation systems for clone management. Most of the work relies on having access to and being able to analyze the history of clone refactoring. However, refactoring cloned code is not equivalent to clone management, as code refactoring can be motivated by goals unrelated to cloning. In this position paper, we introduce a dataset of intentional clone refactoring, which is produced by keywords matching in commit messages within the version control system of Linux kernel. By investigating two important clone evolution scenarios — clone removal and inconsistent changes — in subsystems of Linux kernel, we find that intentional clone refactoring accounts for only a small proportion of all detected clone evolution.

Keywords: Clone Evolution, Code Refactoring, Clone Management

1 Introduction

Software clone management entails detecting, tracking, visualizing, and maintaining clones for developers. Previous research on clone management includes empirical research on clone evolution ([ACD07], [Gö09], [Gö10]) and tool support to track clones ([ZPX+13], [Hot13], [Kos08]).

In previous longitudinal studies of clone management, researchers have primarily employed clone refactoring data to understand clone removal, bug proneness of clones, and clone maintenance patterns. However, for a given clone refactoring instance, it is not always clear if the original developer is aware of the duplication, or if the specific change is made for cloning-related reasons. It is easy to imagine, for example, that the removal of duplication of a clone class could be caused by manual optimization to one clone fragment, instead of a deliberate effort of clone.

In this position paper, we discuss our finding of 348 source code commits that explicitly mention clone management in their associated commit message. After locating code clones (identified by a clone detector) that are changed by these commits (which we call *intentional clone refactoring* in this paper), we report a few general findings of these code clones. We further investigate two clone evolution scenarios — clone fragment removal and inconsistent change in clones — and we find that clones that are associated with intentional clone refactoring account for fewer than 1% of all detected clone changes.

2 Clone Management Data Set

2.1 Recognizing Clone Management in Version Control Logs

The Linux kernel git repository has maintained the version history of Linux kernel since 2005. In analysing commits to this repository, we have identified some commit messages that explicitly discuss cloning-related topics. For example, Fig. 1 shows a commit that describes clone removal its primary activity. Further details of this commit can be found by querying the Linux kernel git repository. During our manual investigation, we have found commit messages that explicitly discuss refactoring of clones, including creating duplications, modifying duplicated code, and the removal of replication.

We use keyword matching to identify intentional clone refactoring in commit messages. To avoid false positives, in this position paper we consider only instances that match the keywords "*duplicated code*" in commit messages. We explicitly do *not* match words such as "*clone*" and its stemming words ("*cloning*", or "*cloned (code)*"), as we have found that in practice these terms exclusively refer to the cloning of git nodes of the version control system, which is obviously unrelated to code cloning as we commonly use the term. By matching word pattern "duplicated code" in commit messages, we identified 348 distinct commits out of a total number of 28,538 commits in Linux kernel git repository; this is likely a lower bound on the amount of cloning discussed in commit messages.

2.2 Clone Detection Results and Clone Management Instances

After recognizing intentional clone refactoring instances from Linux kernel git repository, we incorporate these instances with clone detection results. We apply the Bauhaus *iClones* clone analysis tool on each minor release of Linux kernel (33 releases in total, from Version 2.6.21 to Version 3.12rc), with 20 tokens as the threshold for clone detection. Our goal is to identify the *last* associated clone fragment (identified by clone detector) before the commit is reflected in releases, so that clone refactoring described in commit can also be captured by clone evolution analysis. After generating clone detection result, we match it against the intentional clone refactoring dataset based on location (start/end line number in a file) and version (by running "`git tag --contains`").

Due to the limited scope of this position paper, we restrict our investigations to four major subsystems in the Linux kernel source code base. Table 1 shows the number of matched clone fragments. One surprising finding is that clone fragments involved in intentional clone refactoring are significantly smaller in size, the median of clone fragment tokens for three out of four subsystems are below 35 tokens, while the most common threshold of clone detection we have observed is 50 tokens (for example, in [BPS+07], and [JMSG07]).

3 Clone Evolution in Intentional Clone Refactoring Instances

After matching clone detection result with intentional clone refactoring instances, we study two clone evolution scenarios: clone fragment removal and inconsistent change. The goal of this study is to understand the percentage of intentional clone refactoring over all detectable clone

```
commit 421f38835fe677d8c2e8c25628ae9cd4019653d2

Date:   Tue Feb 28 16:12:44 2012

    USB: serial: whiteheat.c: use module_usb_serial_driver

    This converts the whiteheat.c driver to use the
    module_usb_serial_driver() call instead of having to have a
    module_init/module_exit function, saving a lot of duplicated code.

diff --git  a/drivers/usb/serial/whiteheat.c
            b/drivers/usb/serial/whiteheat.c
    --- a/drivers/usb/serial/whiteheat.c
    +++ b/drivers/usb/serial/whiteheat.c
    @@ -1454,30 +1454,7 @@ out:
```

Figure 1: A commit which indicates intentional clone refactoring in Linux kernel.

Table 1: Clones associated with intentional clone refactoring in Linux kernel

Subsystem	#Clone Fragments	Median of #Token
arch.powerpc	37	29
drivers.infiniband	208	66
drivers.usb	119	35
sound.soc	111	30

Table 2: Clone Fragments Removed in Linux Subsystems

	Intentional Clone Refactoring	Other
arch.powerpc	4	38,801
drivers.infiniband	27	33,179
drivers.usb	37	76,927
sound.soc	12	42,729

evolution instances.

3.1 Clone Fragment Removal

The Bauhaus iClones detects clone evolution by mapping suffix tree-based code changes between consecutive revisions. Details of clone evolution analysis in iClones can be found in [GK09]. In this paper, the clone evolution data provided by iClones is used to indicate clone removal cases. Clones that are clone management in four Linux subsystems are shown in Table 2. Clearly, the intentional removal of cloned code is only a small proportion of entire set of clone removal cases.

Table 3: Clone Fragments with Inconsistent Changes in Linux Subsystems

	Intentional Clone Refactoring	Other
arch.powerpc	6	21,642
drivers.infiniband	33	15,344
drivers.usb	36	51,221
sound.soc	10	65141

3.2 Inconsistent Changes of Clones

Change consistency within a clone class has long been considered as a sign of poor clone management. Similar to clone removal cases, we make direct use of clone change consistency result from Bauhaus iClones. Based on the result shown in Table 3, inconsistent changes in intentional clone refactoring account for only a small proportion of the entire set of inconsistent changes.

4 Discussion

4.1 Implications to Software Clone Research

There is already a line of empirical research on clone management ([Gö09], [Gö10], [ACD07], [Hot13], [WG12]), all of which provide insights into the evolution and maintenance of code clones. However, a clone evolution or maintenance instance, based on findings of clone refactoring, might be made because of a change in functionality — a reason irrelevant to code cloning. In other words, we know little about design decisions that drive refactoring throughout the lifecycle of a clone (from its creation, to its evolution, and to possible removal). Indeed, the result shown in Table 2, and Table 3 suggests that the majority of cloned code refactoring cases are not mentioned in commit messages for four subsystems in Linux kernel.

We argue that intentional clone refactoring can offer strong insights into the practice of cloning in industrial software, and so may be a useful tool in cloning research. Intentional clone refactoring data may indicate features of clone management in practice. For example, according on Table 1, we would have missed a majority of intentional clone refactoring if we had set 50 tokens as the clone minimum length, which is often used in research settings. Likewise, possible patterns in intentional clone refactoring (e.g., ways of removing duplication) seem key to for a better understanding of clone management as practiced; in turn, this may lead to new research opportunities in clone management tool design.

5 Conclusion

In this position paper, we introduce a corpus data of intentional clone refactoring from Linux kernel. By matching keywords in commit messages of version control system, we identify 348 commits with intentional clone refactoring. We select four major subsystems in the Linux kernel and run clone evolution analysis. The result shows that intentional clone refactoring accounts for

only a small share of overall clone removal instances, suggesting that majority of clone removal instances may be caused by reasons other than intentional clone refactoring. This also applies to inconsistent changes of clones. We argue that this observation may affect how we interpret result of clone evolution or clone management. The corpus data of commits with intentional clone refactoring also offers new research opportunity for cloning research.

Acknowledgements: We sincerely thank Rainer Koschke and his research group for providing us with the Bauhaus clone detector, its associated toolset, and aid in their use.

Bibliography

[ACD07] L. Aversano, L. Cerulo, M. Di Penta. How Clones Are Maintained: An Empirical Study. In *Proceedings of the 11th European Conference on Software Maintenance and Reengineering*. CSMR '07, pp. 81–90. IEEE Computer Society, Washington, DC, USA, 2007.

[BPS+07] H. A. Basit, S. J. Puglisi, W. F. Smyth, A. Turpin, S. Jarzabek. Efficient Token Based Clone Detection with Flexible Tokenization. In *The 6th Joint Meeting on European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering: Companion Papers*. ESEC-FSE companion '07, pp. 513–516. ACM, New York, NY, USA, 2007.

[GK09] N. Göde, R. Koschke. Incremental Clone Detection. In *Software Maintenance and Reengineering, 2009. CSMR '09. 13th European Conference on*. Pp. 219–228. 2009.

[Gö09] N. Göde. Evolution of Type-1 Clones. In *Proceedings of the 2009 Ninth IEEE International Working Conference on Source Code Analysis and Manipulation*. SCAM '09, pp. 77–86. IEEE Computer Society, Washington, DC, USA, 2009.

[Gö10] N. Göde. Clone Removal: Fact or Fiction? In *Proceedings of the 4th International Workshop on Software Clones*. IWSC '10, pp. 33–40. ACM, New York, NY, USA, 2010.

[Hot13] K. Hotta. Efficient Code Clone Management based on Historical Analysis and Refactoring Support. 2013.

[JMSG07] L. Jiang, G. Misherghi, Z. Su, S. Glondu. DECKARD: Scalable and Accurate Tree-Based Detection of Code Clones. In *Proceedings of the 29th International Conference on Software Engineering*. ICSE '07, pp. 96–105. IEEE Computer Society, Washington, DC, USA, 2007.

[Kos08] R. Koschke. Frontiers of software clone management. In *Frontiers of Software Maintenance, 2008. FoSM 2008.* Pp. 119–128. 2008.

[WG12] W. Wang, M. W. Godfrey. We have all of the clones, now what? Toward integrating clone analysis into software quality assessment. In *Software Clones (IWSC), 2012 6th International Workshop on*. Pp. 88–89. 2012.

[ZPX+13] G. Zhang, X. Peng, Z. Xing, S. Jiang, H. Wang, W. Zhao. Towards Contextual and On-Demand Code Clone Management by Continuous Monitoring. In *Proc. ASE*. Pp. 497–507. IEEE, 2013.