Electronic Communications of the EASST
Volume 63 (2014)

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

How We Know the Practical Impact of Clone Analysis

— Position Paper —

Norihiro Yoshida, Eunjong Choi , Yuki Yamanaka , Katsuro Inoue

5 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

How We Know the Practical Impact of Clone Analysis

Norihiro Yoshida[1], **Eunjong Choi** [2], **Yuki Yamanaka** [2], **Katsuro Inoue** [2]

[1] yoshida@is.naist.jp
Graduate School of Information Science
Nara Institute of Science and Technology, Japan

[2] ejchoi, y-yuuki, inoue@ist.osaka-u.ac.jp
Graduate School of Information Science and Technology
Osaka University, Japan

Abstract: In order to develop and improve clone analysis techniques for industrial application, it is necessary to know about how those techniques provide impacts on clone management in industry. In this position paper, we discuss approaches to observing the practical impact of clone analysis on the basis of our experience in applying clone analysis into an industrial development process.

Keywords: Clone management, Industrial experience, Refactoring

1 Introduction

The impact of software analytics is generating a lot of discussions in the software engineering community [NII13, MZ13b, MZ13a]. Although a lot of techniques have been proposed on software analytics so far, how those techniques provide positive impacts is still unclear.

The clone research community has provided a greater number of industrial case studies on clone analysis techniques compared to other analytics research fields [JDHW09, HG13, YHKI12]. However, most of those studies simply applied clone analysis techniques to industrial source code, and only a few studies have been done on the application of those techniques into an industrial development process. It is clearly difficult to confirm the practical impact of applying clone analysis techniques without any application of those techniques into an industrial development process. Mining Software Repositories (MSR) allows researchers to know the evolution of code clones in version archives[KSNM05, GK11, SRS13]. However, it does not tell them whether or not the evolution was intentionally achieved by a developer who was inspired by the result of clone analysis. For example, MSR is insufficient to distinguish whether clone analysis techniques helped a developer to notice code clones to be maintained or the developer noticed them without any clone analysis technique.

In this position paper, we introduce our experience in observing the practical impact of clone analysis during the application of the analysis into a development process in NEC Corporation. During the application, we successfully confirmed that our clone change management system helped a project manager to notice newly-appeared code clones to be maintained. Then, we discuss future challenges for observing the practical impact.

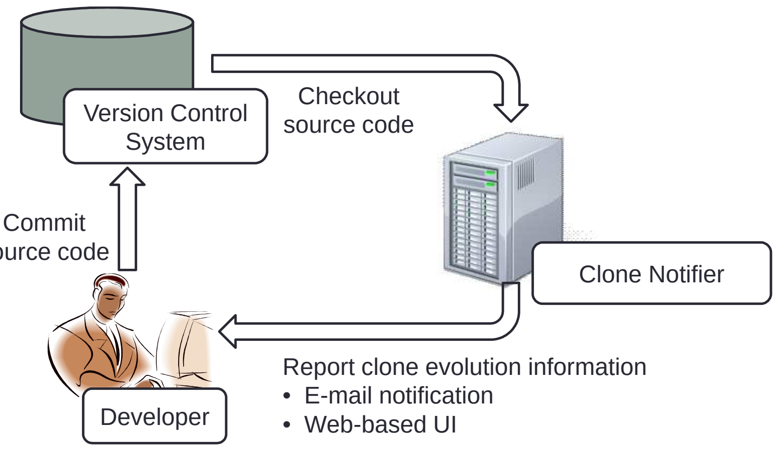

Figure 1: An Overview of Clone Notifier

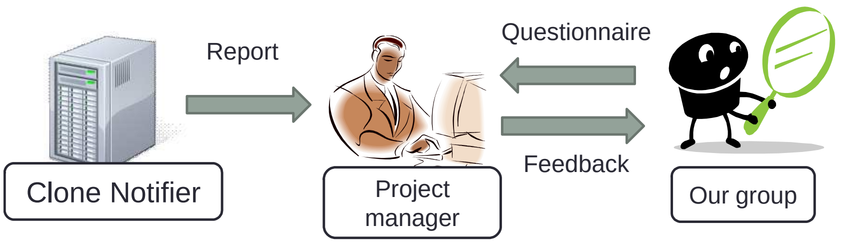

Figure 2: An Overview of the Case Study

2 Industrial Experience with Clone Change Notification System

In this section, we introduce the experience with 40-day application of Clone Notifier [YCY+13] into a development process in NEC Corporation. Figure 1 shows an overview of Clone Notifier. It is a clone change notification system for notifying creation and change of code clones. It regularly identifies newly-appeared and changed clones by analyzing SVN commits and using CCFinder[KKI02], and then notifies them to developers by e-mail and a web-based UI. We used the default setting of CCFinder. In the default setting, the minimum token length of the detected code clones is 30.

The target system consists of approximately 350 files and 12 KLOC written in Java. The duration of the application was about 40 days, from December 2011 to January 2012.

To observe the practical impact of the clone analysis technique, we used questionnaires to the project manager of the development team. Figure 2 shows an overview of the questionnaires. At first, we asked him whether he discovered at least one unintentionally-developed clone, and then asked him how the development team should maintain the unintentionally-developed clone. We showed three options for the second question (i.e., refactoring, write a source code comment to denote the existence of the clone, leave the clone as it is).

2.1 Application result

During 40 days application, Clone Notifier notified overall 119 newly-appeared clone sets. Between 119 clone set of newly-appeared clone set, the project manager recognized 10 clone sets as clone sets should be refactored, and a clone set should be noted in a source code comment.

In the questionnaires, he told us that his criteria whether or not a newly-appeared clone set should be maintained (i.e., refactored, noted) are the amount of code clones that can be reduced in the clone set and the possibility of editing the clone set in the future maintenance, and the criterion whether a newly-appeared clone set should be refactored or noted is the cost of refactoring. The noted clone set includes a number of the name differences between code clones. Therefore, he regarded merging those differences and performing a test to verify the merging as time-consuming task.

Two of the ten clone sets was merged into a single function during the 40 days, respectively. The other eight clone sets were designated as refactoring candidates that will be merged during next maintenance project.

2.2 Manual observation of newly-appeared clone sets

As an ex-post analysis, we investigated the characteristics of clone sets recognized as refactoring candidates by the experienced project manager at NEC. The aim of the analysis is data collection for the development of a technique to recommend refactoring candidates from all newly-appeared and changed clones. The recommendation is able to help developers to reduce the cost of finding clone sets should be merged into a single module.

We manually checked the differences between the 10 clone sets should be merged and the other 109 clone sets. As a result, we learned interesting insights about the clone sets that should be merged.

2.2.1 Code clones Introduced without code addition

As the first insight, in the case of clone sets that were newly-appeared by adding new code, the project manager frequently recognized them as ones should be merged. On the other hand, clone sets were sometimes accidentally created by only the replacement or the deletion of statements. In other words, even if no line is added to a code fragment, it sometimes became a code clone in a clone set together with other code fragments when at least one character is changed in it. In such case, the project manager mostly decided to leave those duplicates as it is.

From our observation of the 119 clone sets, we found that only coding idioms (e.g., programming or API/library specific idioms) are involved in clone sets that were newly-appeared by only the replacement or the deletion of statements. Basically, such idioms are difficult to merge or have an overall positive effect on maintenance and development [KG06, KG08] therefore they should be eliminated from refactoring candidates.

According to this observation, we eliminated clone sets that were newly-appeared by only the replacement or the deletion of statements from the 119 clone sets. The result shows that the elimination not only left the all of clone sets that should be refactored but also reduced the number of the number of 119 newly-appeared clone sets by approximately 86%(16/119).

2.2.2 Syntactically incomplete clone sets

As the second insight, code sets include whole parts of loop or branch statements were considered as ones should be merged. Meanwhile, the project manager rarely recognized clone sets include only parts of loop or branch statements as ones should be merged because it is difficult to merge syntactically incomplete clone sets.

According to this observation, we eliminated syntactically incomplete clone sets from the 119 clone sets. The result shows that the elimination not only left the all of clone sets should be refactored but also reduced the number of 119 newly-appeared clone sets by approximately 90%(12/119). Note that this does not include the previous reduction.

3 Challenges

In this section, we discuss future challenges for observing the practical impact of clone analysis.

At first, the clone research community should develop a collection of common and disciplined approaches for the observation to share and generalize study results presented by different researchers. We believe that our industrial experience introduced in Section 2 can be a good starting point to discuss a collection of common and disciplined approaches for the observation.

Secondly, it is strongly needed to apply various kinds of clone analysis techniques into industrial development processes because it is promising to give a clue to the development of clone analysis techniques. As introduced in Section 2, applying clone analysis into development process tells us what kinds of clones should be notified to developers for what kinds of maintenance activities. For example, the industrial case study in Section 2 tells us newly-appeared clones should be notified to developers who perform refactoring.

Finally, it is also important for the improvement of clone analysis techniques to perform ex-post analysis after the application of those techniques into development process. As introduced in Section 2, our ex-post analysis tells us that more sophisticated techniques are needed for efficient clone notification.

Acknowledgements: This work was supported by JSPS KAKENHI Grant Numbers 25220003, 21240002.

Bibliography

[GK11] N. Göde, R. Koschke. Frequency and Risks of Changes to Clones. In *Proceedings of the 33rd International Conference on Software Engineering*. ICSE '11, pp. 311–320. ACM, New York, NY, USA, 2011.
doi:10.1145/1985793.1985836
http://doi.acm.org/10.1145/1985793.1985836

[HG13] J. Harder, N. Gde. Cloned code: stable code. *Journal of Software: Evolution and Process* 25(10):1063–1088, 2013.

doi:10.1002/smr.1551
http://dx.doi.org/10.1002/smr.1551

[JDHW09] E. Juergens, F. Deissenboeck, B. Hummel, S. Wagner. Do code clones matter? In *Proc. of ICSE*. Pp. 485–495. 2009.

[KG06] C. Kapser, M. W. Godfrey. ”Cloning Considered Harmful” Considered Harmful. In *WCRE*. Pp. 19–28. 2006.

[KG08] C. J. Kapser, M. W. Godfrey. ”Cloning considered harmful” considered harmful: patterns of cloning in software. *Empir Software Eng* 13(6):645–692, 2008.

[KKI02] T. Kamiya, S. Kusumoto, K. Inoue. CCFinder: a Multilinguistic Token-Based Code Clone Detection System for Large Scale Source Code. *IEEE Trans. Softw. Eng.* 28(1):654–670, 2002.

[KSNM05] M. Kim, V. Sazawal, D. Notkin, G. C. Murphy. An empirical study of code clone genealogies. In *Proc. of ESEC/FSE*. Pp. 187–196. 2005.

[MZ13a] T. Menzies, T. Zimmermann. The Many Faces of Software Analytics. *IEEE Software* 30(5):28–29, 2013.
doi:http://doi.ieeecomputersociety.org/10.1109/MS.2013.114

[MZ13b] T. Menzies, T. Zimmermann. Software Analytics: So What? *IEEE Software* 30(4):31–37, 2013.
doi:http://doi.ieeecomputersociety.org/10.1109/MS.2013.86

[NII13] NII Shonan Meeting on Software Analytics: Principles and Practice. 2013.
http://www.nii.ac.jp/shonan/blog/2012/11/19/software-analytics-principles-and-practice/

[SRS13] R. K. Saha, C. K. Roy, K. A. Schneider. gCad: A Near-Miss Clone Genealogy Extractor to Support Clone Evolution Analysis. In *Software Maintenance (ICSM), 2013 29th IEEE International Conference on*. Pp. 488–491. 2013.
doi:10.1109/ICSM.2013.79

[YCY$^+$13] Y. Yamanaka, E. Choi, N. Yoshida, K. Inoue, T. Sano. Applying Clone Change Notification System into an Industrial Development Process. In *ICPC*. Pp. 199–206. 2013.

[YHKI12] N. Yoshida, Y. Higo, S. Kusumoto, K. Inoue. An Experience Report on Analyzing Industrial Software Systems Using Code Clone Detection Techniques. In *Software Engineering Conference (APSEC), 2012 19th Asia-Pacific*. Volume 1, pp. 310–313. 2012.
doi:10.1109/APSEC.2012.98