Clone Evolution Revisited

Nils Göde
University of Bremen
Bremen, Germany
nils@informatik.uni-bremen.de

Marcus Rausch
Debeka-Group
Koblenz, Germany
Marcus.Rausch@debeka.de

Abstract

The majority of studies that analyze clone evolution is limited to open-source software. Although providing important insights, it is unsure whether findings can be transferred to industrial contexts. We conducted a study on clone evolution in industrial systems to complement the findings of our previous study on clone evolution, which was limited to open-source systems. In this paper, we present the results from the industrial context and compare them to our previous findings based on open-source software.

1 Introduction

Duplicated fragments of source code—code clones—are an inevitable property of every software system. Certain types of clones negatively affect software maintenance as the effort needed to understand and change the code increases in the presence of code clones. Furthermore, clones increase the risk of introducing defects by inconsistent changes to parts of the code that are meant to evolve identically. Hence, it is important to analyze the evolution of clones to identify the clones that indicate potential threats.

To better understand the general phenomena of cloning, we conducted an extensive study on clone evolution [2]. Although the study analyzed nine different systems, all of them were open-source systems. Thus, we cannot tell whether the results are transferable to industrial projects that differ in the development and maintenance context. In this paper, we present our analysis of clone evolution in industrial systems and compare the results to those of our previous study. To ensure comparability, we use the same study procedure. In particular, we answer the following research questions:

Q 1—*How does the clone ratio change over time?*

Q 2—*How long do cloned fragments exist?*

Q 3—*What is the ratio between consistent and inconsistent changes?*

2 Related Work

There exists a variety of studies of different aspects of clone evolution, e.g., change of the clone ratio [7], lifetime of clones [4], and consistency of changes to clones [1, 5]. However, all of these studies were based on open-source systems. To the best of our knowledge, only Laguë and colleagues [6] have analyzed clone evolution in industrial software. They found that the overall ratio of clones remained stable.

3 Subject Systems

For our study, we used three industrial systems developed by the Debeka-Group. The Debeka-Group offers a variety of insurance and financial services and is one of the top-ten companies in the insurance and home savings business. The systems we have studied are written in COBOL and each has a history of more than ten years. Systems *zy-zg* and *zy-zp* provide functionality common to different services, whereas the system *kv-kl* is dedicated to health insurance.

For each system, we analyzed clone evolution over 125 versions with an interval of one week. The first version is from 10 November 2007, the last from 27 March 2010. Earlier versions were not available due to a change in version control mechanisms. Clone detection and related techniques have not been used during the maintenance process. The size of the first and last version of each system is given in Table 1.

4 Study Procedure

We used our incremental clone detection tool *iClones* to detect clones in each version and map those of consecutive versions [3]. We limited the detection to type-1 clones, because—according to our experience—these ensure high precision. Contrary to our previous study, we used only a minimum clone length of 100 tokens, since COBOL programs contain longer duplicated sequences that are unavoidable and hence, 50 tokens minimum length results in too many false positives, as our prestudy has shown.

To answer Q 1 we compared the increase in source code size to the increase or decrease in the number of cloned fragments. This allows to identify trends that indicate whether systems are increasingly threatened by code clones or not.

Exploiting the mapping of clones between versions, we measured the lifetime of each fragment to answer Q 2. The lifetime of a fragment starts when it has no ancestor and ends when there is no descendant. For details on how ancestors and descendants are deter-

Table 1: Subject Systems

	Size [KSLOC] (Q 1)			Fragments (Q 1)				
System	First	Last	Δ [%]	First	Last	Δ [%]	Lifetime [#V.] (Q 2)	*icratio* [%] (Q 3)
kv-kl	530	623	17.68	5127	6348	23.82	87	75.4
zy-zg	148	153	3.32	171	155	−9.36	125	92.1
zy-zp	422	448	6.10	1316	1238	−5.93	47	80.7

mined, please refer to our previous work [2, 3].

Regarding Q 3, we compared the number of inconsistent changes to all changes affecting clones. In particular, we count the number of cloned fragments that are part of inconsistently changed clone classes to the number of fragments contained in all changed clone classes. Let this be the *icratio*.

5 Results

The detailed results are given in Table 1. Regarding Q 1, all three systems have increased in size during the period of study. While *zy-zg* and *zy-zp* increased only slightly, *kv-kl* grew more than 17%. Interestingly, the number of fragments decreased in *zy-zg* and *zy-zp* although the source size increased. In *kv-kl*, however, the number of fragments increased by almost 24%. Comparing the change in size and the change in number of fragments, the clone ratio decreased in *zy-zg* and *zy-zp*, whereas it increased in *kv-kl*.

The answer to Q 2 is different for the three systems. The median of the fragments' lifetimes measured in versions is given in Table 1. These values have to be regarded as a lower bound since many fragments likely existed before the window of analysis and might continue beyond the end of our analysis period. Fragments in *zy-zg* have the highest life expectancy of 125 versions, which equals the total period of study. Fragments in *kv-kl* exist for 87 versions whereas fragments in *zy-zp* last only 47 versions on average.

Regarding Q 3, clearly more fragments are affected by inconsistent changes than by consistent changes. For all three systems, the *icratio* is above 75%, thus inconsistent changes dominate. The highest *icratio* was observed for *zy-zg*. Of all fragments contained in changed clone classes, 92% are associated with inconsistent changes.

6 Discussion

The number of fragments dropped in *zy-zg* and *zy-zp* although both systems grew. In *kv-kl*, however, the number of fragments increased stronger than the size of the system. Following from the answer to Q 1, we cannot say that cloning naturally becomes worse over time without employing clone management. Nevertheless, some systems seem to be particularly susceptible to an increasing impact of clones. We also observed these differences in our previous study on open-source systems.

The lifetime of clones is different for the three systems, yet fragments exist on average for at least 47 versions—almost one year. Again, the answer to Q 2 supports our previous findings and shows that clones in long-lived systems are not volatile at all.

Consistent with our previous result, the majority of clones are not changed during the period of study. We assume this to be a general property of mature systems. Regarding Q 3, the results are different from our previous study, where we have found the highest *icratio* to be 74.2%. All three systems in this study have a higher *icratio* than in our previous studies on open-source systems.

We are well aware, that our findings may—to a certain degree—depend on the differences between the COBOL programming language and the languages we have analyzed in our previous study. Further inspection of the inconsistent changes is required to tell whether the higher degree of inconsistencies originates from the use of COBOL or particular characteristics of industrial projects.

7 Conclusion

On the one hand, we have found commonalities between open-source and industrial systems. Clones are rarely changed and not volatile, as most of them exist for more than a year. On the other hand, the ratio of inconsistent changes to code clones is clearly higher for the three industrial systems than for the open-source systems. In general, the conclusion from our previous study, that clone evolution is considerably different for individual systems and hard to generalize, also applies to industrial systems.

References

[1] L. Aversano, L. Cerulo, and M. Di Penta. How clones are maintained: An empirical study. In *CSMR*, 2007.

[2] N. Göde. Evolution of type-1 clones. In *SCAM*, 2009.

[3] N. Göde. Mapping code clones using incremental clone detection. *Softwaretechnik-Trends*, 29(2), 2009.

[4] M. Kim, V. Sazawal, D. Notkin, and G. C. Murphy. An empirical study of code clone genealogies. In *ESEC/FSE*, 2005.

[5] J. Krinke. A study of consistent and inconsistent changes to code clones. In *WCRE*, 2007.

[6] B. Laguë, D. Proulx, J. Mayrand, E. Merlo, and J. Hudepohl. Assessing the benefits of incorporating function clone detection in a development process. In *ICSM*, 1997.

[7] S. Livieri, Y. Higo, M. Matsushita, and K. Inoue. Analysis of the linux kernel evolution using code clone coverage. In *MSR*, 2007.