Experience of Finding Inconsistently-Changed Bugs in Code Clones of Mobile Software

Katsuro Inoue†, Yoshiki Higo†, Norihiro Yoshida†, Eunjong Choi†, Shinji Kusumoto†,
Kyonghwan Kim‡, Wonjin Park‡, and Eunha Lee‡
†*Osaka University* ‡*Samsung Electronics Co.*
Osaka, Japan *Suwon, South Korea*
{*inoue, higo, n-yosida, ejchoi, kusumoto*}*@ist.osaka-u.ac.jp*
{*kyonghwan73.kim, wj23.park, leeeunha*}*@samsung.com*

***Abstract*—When we reuse a code fragment, some of the identifiers in the fragment might be systematically changed to others. Failing these changes would become a potential bug in the copied fragment. We have developed a tool *CloneInspector* to detect such inconsistent changes in the code clones, and applied it to two mobile software systems. Using this tool, we were effectively able to find latent bugs in those systems.**

***Keywords*-Inconsistent Change, Unchanged Ratio, Bug Candidate**

I. Introduction

Software systems for mobile phone (mobile software) are becoming huge and complex, and debugging and maintaining them are getting difficult and expensive.

A mobile software system needs to adapt its features to various country/area constraints, and so many code clones are generated and embedded in the system. Code clones might introduce their unique bugs.

Consider a type-2 code clone pair $X = aibic$ and $Y = ajbic$, where a, b, c, i, and j are tokens, and i and j are specifically identifiers. Y was copied from X, and all occurrences of identifier i in X must be changed to j in Y. However, the second occurrence of i is not changed to j and it might cause a failure. We call such an unintentional and inconsistent identifier change *an inconsistently-changed bug*. Inconsistently-changed bug is one of the potential risks for complex software systems such as recent mobile software.

Krinke has analyzed consistency of code clone changes over revisions of OSS, and found inconsistent change cases without later consistent changes [3]. Li et al. have developed a tool named CP-Miner to detect inconsistently-changed bugs using the clone detection technique based on the frequent subsequence mining [4]. It seems that CP-Miner would provide basic features for our requirements, but we had to customize the tool to fit to the development environment of Samsung.

We had already developed a prototype tool to detect inconsistently-changed bugs using code clone detector CCFinder and its post processor [1], [2], [5]. However, this prototype tool was unable to handle a large size input, and it did not have industry-required strength.

We have restructured and modified various parts of the prototype tool, and have built a tool named *CloneInspector*.

In this paper, we will show an overview of CloneInspector, and present our experience of applying CloneInspector to Samsung's large mobile software.

II. CloneInspector

Figure 1 shows the process of CloneInspector.

1) First, code clones in the input source files are detected by code clone detector CCFinder [2]. The positions of code clones are generated.
2) Using the positions, code fragments for the detected code clones are tokenized. At the same time, the occurrences of identifiers are examined.
3) For each clone pair (S, T), mapping of every identifier is checked. If there is an identifier m with multiple occurrences in S, and m is mapped to more than one identifiers such as n and o, then we say that clone pair (S, T) is *inconsistently changed* (either n or o may be the same as m.). Only inconsistently changed clones are extracted as the initial bug candidates here.
4) *Unchanged Ratio* for an identifier m in S is the ratio of the m's occurrences in T over all m's occurrences in S. For example, if S has three m's occurrences and they are mapped to two n's occurrences and one m's occurrence in T, then the unchanged ratio for m in S is 0.33. An unchanged ratio takes a value between 0 to 1, and a smaller value close to zero means that a small number of the original identifier's occurrences remain unchanged. This would suggest the existence of possible bugs. In this step, the unchanged ratios for all identifiers in inconsistently changed clones are computed.
5) The identifiers with unchanged ratios larger than the criterion are removed from the initial bug candidates. Also, the identifiers which are mapped to more than two different identifiers are removed since we would consider that those are intentionally mapped to many different identifiers including the original one. The

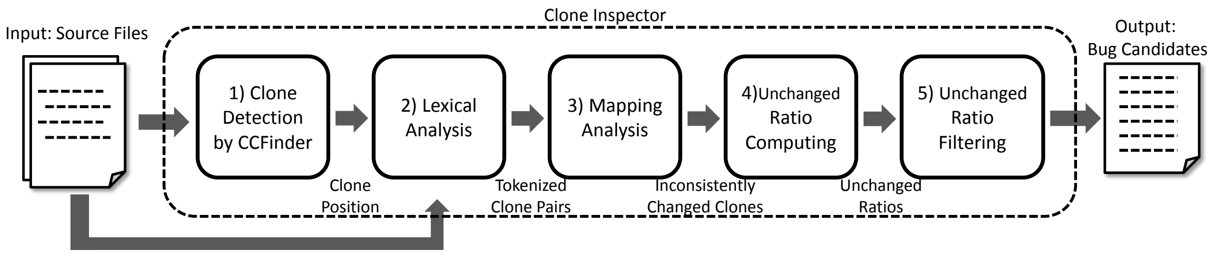

Figure 1. Process of CloneInspector

Table I
TARGET MOBILE SOFTWARE SYSTEMS AND APPLICATION RESULTS

	System A	System B
Feature	Communication	Application
Language	C	C
Size (LOC)	4,275,952	136,554
Clone Set	38,192	4,053
Reported Bug	63	5
Validated Bug	25	1
Exec. Time (CCFinder)	300sec	40sec
Exec. Time (Others)	143sec	4sec

remaining identifiers are reported as the (final) bug candidates.

CloneInspector has been implemented mostly in Java (except for CCFinder), and it can handle C, C++, and Java as its input languages. We have validated it's scalability with about 10M LOC input files.

III. APPLICATIONS

We have applied CloneInspector to two mobile software systems of Samsung. The various characteristics of those systems and the execution results are shown in Table I.

The execution environment of these applications was 2.67GHz Intel Core i5 CPU with 4.0GB main memory. The minimum token length of CCFinder was 50, and the maximum unchanged ratio criterion was 0.3.

In the case of System A, about 4M LOC target program has been inspected, and 63 bug candidates were reported. Among these candidates, 25 have been identified as true bugs by our manual inspection.

In the case of System B, we have found 1 true bug from 5 reported candidates.

All of these bugs were not known bugs. Therefore, those bugs were reported to the development section, and they have been fixed in the newer versions.

The overall performance of CloneInspector is very fine, because it quickly reports small number of bug candidates. The execution times are several minutes even for fairly large systems.

We have examined the output bug candidates with unchanged ratio 0.4. By this setting, CloneInspector reported 193 bug candidates for System A and 7 for System B, which are more than the cases of 0.3. However, all of increased candidates were identified as intentional inconsistencies (not bugs). This would suggest that an upper bound of the practical unchanged ration would be less than 0.4. We will explore other settings to find the best unchanged ratio.

IV. CONCLUSION

In this paper, we have shown our experience of developing CloneInspector and applying it to Samsung's mobile software. The result is very fine, and we are currently using it and will continue to use it in practice.

Current CloneInspector cannot find bugs with the mapping to more than two different identifiers. We will analyze such cases and find a way to detect such bugs.

Now we have a plan to extend CloneInspector to generate more less false positive bug candidates, by introducing various cases of intentional inconsistency. Also, we will add new Web-based GUI to share the output report with many developers.

ACKNOWLEDGEMENT

This work has been fully supported by Samsung Electronics Co. We are grateful to many colleagues, including Yasuhiro Hayase and Yong Lee Yii, who have contributed to the development of CloneInspector.

REFERENCES

[1] Y. Hayase, Y. Yii, K. Inoue: "A Criterion for Filtering Code Clone Related Bugs", Proceedings of International Workshop on Defects in Large Software Systems (DEFECTS 2008), Seattle, WA, pp.37-38, July 2008.
[2] T. Kamiya, S. Kusumoto, K. Inoue, "CCFinder: A Multilinguistic Token-Based Code Clone Detection System for Large Scale Source Code", IEEE Trans. on Software Engineering, Vol. 28, No. 7, pp. 654-670, July 2002.
[3] J. Krinke, "A Study of Consistent and Inconsistent Changes to Code Clones", 14th WCRE, pp. 170-178, Vancouver, Canada, October 2007.
[4] Z. Li, S. Lu, S. Myagmar, and Y. Zhou, "CP-Miner: Finding Copy-Paste and Related Bugs in Large-Scale Software Code", IEEE Trans. Software Engineering, 32(3), pp. 176-192, March 2006.
[5] Y. Yii, Y. Hayase, M. Matsushita, and K. Inoue, "Token Comparison Approach to Detect Code Clone-Related Bugs", Technical Paper of IEICE, SS2007-57 75, Vol.107, No.505, pp.37-42, March 2008.