The Journal of Systems and Software 134 (2017) 105–119

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Predicting change consistency in a clone group

Fanlong Zhang[a,1], Siau-cheng Khoo[b,2], Xiaohong Su[a,*]

[a] School of Computer Science and Technology, Harbin Institute of Technology, Harbin, China
[b] School of Computing, National University of Singapore, Singapore

ARTICLE INFO

Article history:
Received 24 October 2016
Revised 21 August 2017
Accepted 29 August 2017
Available online 31 August 2017

Keywords:
Software reuse
Clone maintenance
Code clones
Consistency-requirement prediction
Bayesian network
Clone attributes

ABSTRACT

Code cloning has been accepted as one of the general code reuse methods in software development, thanks to the increasing demand in rapid software production. The introduction of *clone groups* and *clone genealogies* enable software developers to be aware of the presence of and changes to clones as a collective group; they also allow developers to understand how clone groups evolve throughout software life cycle. Due to similarity in codes within a clone group, a change in one piece of the code may require developers to make *consistent change* to other clones in the group. Failure in making such consistent change to a clone group when necessary is commonly known as "clone consistency-defect", which can adversely impact software reusability.

In this work, we propose an approach to predict the need for making *consistent change in clones within a clone group* at the time when changes have been made to one of its clones. We build a variant of clone genealogies to collect all consistent/inconsistent changes to clone groups, and extract three attribute sets from clone groups as input for predicting the need for consistent clone change. These three attribute sets are code attributes, context attributes and evolution attributes respectively. Together, they provide a holistic view about clone changes. We conduct experiments on four open source projects. Our experiments show that our approach has reasonable precision and recall in predicting whether a clone group requires (or is free of) consistent change. This holistic approach can aid developers in maintaining clone changes, and avoid potential consistency-defect, which can improve software quality and reusability.

1. Introduction

A *clone fragment* (or simply called a *clone*) is generally referred to as a piece of code fragment that is "similar" to another piece of code fragment; the notion of "similarity" between two code fragments is typically defined at textual or syntactical level (Koschke, 2007). "Copy-and-paste" operation is the most noticeable way for physically reusing existing code from software, and it can introduce abundant clone fragments. The presence of clones in software has given rise to the question of whether clones can adversely affect software quality, starting with Fowler et al. identifying some of the clones as "bad smell" (Fowler and Beck, 1999). Clone research community has since debated on whether changing a set of clones inconsistently may cause software defects, and whether the requirement for changing clones consistently may lead to extra maintenance cost. If clone fragments in a group of clones need to be changed consistently and developers forget to do so, it may introduce defects at latter stage of software evolution (Bettenburg et al., 2009; Juergens et al., 2009). On the other hand, when consistent change within a clone group is not required, developers might unnecessarily spend time on verifying and attempting to maintain clone consistency, resulting in additional software maintenance overhead (Aversano et al., 2007; Barbour et al., 2011).

This work is an extension of our conference paper (Zhang et al., 2016), which outlines a *predictive model that warns software developers about the need to perform consistent change in clones,* so as to reduce clone maintainability cost in specificity, and improve software maintainability in general. The extension here includes experimental details as well as the inclusion of another software repository as experiment subject. Moreover, we extend the technique to include prediction of clone changes which do NOT require consistent change to the corresponding clone group. Thus, in this work, we develop a more holistic approach which *predicts whether consistent change is needed for a clone group when one of clone fragments in the group has been modified.* Specifically, when a developer mod-

* Corresponding author.
E-mail addresses: hitzhangfanlong@gmail.com (F. Zhang), khoosc@nus.edu.sg (S.-c. Khoo), sxh@hit.edu.cn (X. Su).

[1] Fanlong Zhang is the main author, and most of the work was done by him.

[2] The major part of this work was done when the first author was on a PhD internship at National University of Singapore.

http://dx.doi.org/10.1016/j.jss.2017.08.045
0164-1212/

ifies a piece of code which is a clone of other code, our developed model will make its prediction, and offer two possible warnings to developers:

1. When similar changes are indeed required for **at least one** other clones in a clone group, we say that the clone group satisfies the *clone consistency-requirement*. If this requirement is predicted, our model will alarm the developer, and appropriate management action can be taken to avoid consistency-defect. Although this leads to an increase in software maintenance cost, it reduces the risk for clone-consistency-defect.
2. When **none** of the clones in the clone group requires consistent change, we say that the clone group are *consistency-free*. If this requirement is predicted, our model will inform the developer, who can then change the clones freely with more confidence. This in turns saves unnecessarily time on verifying consistency.

A related work done in this direction of clone consistency-requirement prediction, which has inspired the current work, was conducted by Wang et al. (2012, 2014). In that work, they define a code cloning operation as "consistency-maintenance-requirement" if its generated code clones experience consistent changes in software evolution history. They aim to automatically predict whether a code cloning operation requires consistency-maintenance *at the time* when a copy-and-paste operation is performed. They employ Bayesian network machine learning technique (Friedman et al., 1997) to develop and train a prediction model. Their predictor is built with the following two categories of inputs: (1) the syntactic characteristics of the code and its copy-and-paste counterpart and (2) the physical context of the code and its copy-and-paste counterpart. Tested on two open source projects and two large-scale internal projects, they show that their predictor is able to recommend developers to perform more than 50% of cloning operations with a precision of at least 94% in these four subjects; in addition, it is also able to avoid 37% to 72% of consistency-maintenance-required code clones by warning developers on only 13% to 40% code clones.

While Wang et al. aim to perform prediction at copy-and-paste time, we perform prediction at *almost any time* in software life cycle when a clone has been modified. Our technique can thus be applied to existing clones in an established project, rather than new clones formed (via copy-and-paste). To achieve that, we need to be aware of the presence of clone group to which the modified code belongs. A *clone group* is a group of clones within a piece of software which are known to be similar by some similarity measures. In order to train a predictor, it is natural to investigate the evolution of clone groups during software evolution. To this end, we adapt the notion of *clone genealogy* as nicely explained by Kim et al. (2005). A clone genealogy describes the evolution of clones, and defines various *clone patterns* to describe how clones in a group have been changed from the earlier version of the project. We hypothesize that *how a clone had been modified genealogically wrt its clone group has an impact on the prediction if the clone group requires consistent change in future*. We thus build our predictor based on three categories of inputs, two of them have been adapted from the work by Wang et al. (2012, 2014), and the last one captures the characteristics of clone genealogy, called *evolution attributes*. This combination of three attributes provides a holistic view on clone groups; the presence of evolution attributes enables the predictor to be customized to individual software repository. We develop and construct, via WEKA (Hall et al., 2009), a Bayesian network as the predictor, and experiment on its predictive power on three software projects. Our experiments show that: the predictor performs reasonably well with stable precision and recall for both its prediction for clone consistency-requirement and consistency-free, with precision ranges between 70% to 80%, and its recall between 63% and 83%. In addition, each of the attribute sets contributes positively in its own way to the predictive power, and an absence of any of these attribute sets can adversely affect the recall ability of the predictor.

The contributions of this paper are as follows:

1. We propose an approach to predict the need for consistent change in a clone group arising from the occurrence of a clone change.
2. We identify a new set of attribute for prediction based on information related to clone genealogies. The results show that this set of attributes has positive impact on the recall ability of the predictor.
3. We demonstrate the feasibility of this prediction via an evaluation on four open source projects. The results show that our approach can predict consistent change in clone group effectively with good precision and reasonable recall, and can help improve software reusability through predictive clone maintenance.

This paper is organized as follows: Section 2 discusses related works.We give a brief introduction of code clone research and defect prediction. Preliminaries is provided in Section 3. There, we explain code clone types, clone genealogy and clone changes. And also give a example of consistent change from real world. Section 4 details our approach to consistent clone change prediction. We provide the detail of implementation in Section 5. Section 6 describes our evaluation through experimentation on four projects. Section 7 discusses threats to validity. We conclude and point to future work in Section 8.

2. Related works

Ever since code clones have been identified as "bad smell" (Fowler and Beck, 1999), there have been discussions over its harmfulness to software. Proponents for clones being harmful opine that their existence can lead to software defect and incur additional maintenance effort. To this end, Lozano and Wermelinger, studying the changeability of code clones, show that the maintenance effort of changing a method may increase significantly when the method has a clone (Lozano and Wermelinger, 2008). In addition, Barbour et al. investigate unsynchronized changes to clone pairs which are re-synchronized at later stage – called late propagation — and show that such pattern is related to a high number of defects (Barbour et al., 2011). What's more, Bakota et al. propose a definition of "clone smell" to determine whether clones are related to software defects (Bakota et al., 2007). Those studies can provide the evidence that code clones may hold certain characteristics that are harmful to the enclosing software. However, opponents for clones being harmful have different research results. For instance, Rahman et al. study the relationship between cloning and defect proneness on open source projects, and find that clones may be less defect prone than non-cloned code (Rahman et al., 2012). This study appears diametrically opposite to the views on clone harmfulness. Actually, the harmfulness of code clones tend to have a strong subjectivity, which usually depends on the perspective of individual researcher. Therefore, there are also others researchers who adopt a neutral attitude towards clone harmfulness. For instance, Kapser and Godfrey describe several clone patterns, discuss their pros and cons, and conclude that there are situations where clone seems to be a reasonable or even beneficial design option (Kapser and Godfrey, 2006).

Regardless of whether the presence of code clones is deemed harmful, "cloning" as a program development activity will persist due to the manner in which general software developers sometime develop codes (out of convenience). Clone analysis, especially on clone evolution and its change pattern, plays an important role in understanding these clones. In this regard, Kim et al.