We Have All of the Clones, Now What? Toward Integrating Clone Analysis into Software Quality Assessment

Wei Wang and Michael W. Godfrey
David R. Cheriton School of Computer Science
University of Waterloo, Waterloo, ON, CANADA
Email: {w65wang, migod}@uwaterloo.ca

***Abstract*—Cloning might seems to be an unconventional way of designing and developing software, yet it is very widely practised in industrial development. The cloning research community has made substantial progress on modeling, detecting, and analyzing software clones. Although there is continuing discussion on the real role of clones on software quality, our community may agree on the need for advancing clone management techniques. Current clone management techniques concentrate on providing editing tools that allow developers to easily inspect clone instances, track their evolution, and check change consistency. In this position paper, we argue that better clone management can be achieved by responding to the fundamental needs of industry practitioners. And the possible research directions include a software problem-oriented taxonomy of clones, and a better structured clone detection report. We believe this line of research should inspire new techniques, and reach to a much wider range of professionals from both the research and industry community.**

I. Introduction

The software cloning research community has reached significant achievements in many dimensions. Several clone detectors are available to perform highly accurate clone detection on large software systems. We have several tools that can model and detect the evolution of clones, and we have a series of empirical studies on the evolution of clones. All these achievements in cloning analysis are leading to better understanding of cloning as a development practice.

Cloning may cause problems in software. However, just as other “symptoms” of software systems — such as software failures or code bloats — fixing systems based on the description of a “symptom” is never a simple task. Determining the root cause of a bug can turn into a time-consuming puzzle. Fixing a bug may cause unexpected side-effects. Many factors contribute to this reality. Despite the ever-increasing size and complexity of many software systems, the lack of cost-benefit estimation to address these software “symptoms” is often considered as an important factor. This explanation certainly applies well to the “symptom” of cloning.

Harder and Göde [4] reported their experience of clone management. In their story, three developers were involved to simply determine the reason of introducing one clone instance in their system yet no substantial gain of managing that clone could be identified. This story reflects many essential difficulties in clone management. The lack of cost-effectiveness evaluation is one of them, as Harder and Göde pointed out.

We believe there are other challenges in clone management. Note that one clone instance is singled out and studied by Harder and Göde. Cloned code typically takes up to 10% - 13% of the entire code base, a clone report from a large software system is always expected to be long, containing lots of clones, unstructured and unorganized. That is to say, there are several challenges that stop us from delivering an effective clone management solution from a clone detection result.

II. The state-of-art of clone management

Cloning is no longer considered as a solely problematic practice. Developers may have well justified reasons to reuse software designs by cloning, especially for critical systems [1]. Cloning may also be introduced with good design and management intentions [5]. Simply put, cloning can play a positive role on the system’s design and ultimate evolution; and this makes clone management not merely a “hunt-and-kill” game.

Current clone management research focuses on providing editing tools within development environments that ease the inspection of the pair (or set) of a duplicated code chunk and its associated evolution and change consistency. Duala-Ekoko and Robillard [2] designed a clone management tool which used heuristics to create an abstract representation of clones and tracked the evolution of clones. The cloning analysis results — including clone evolution and consistency — are provided to invite developers to make maintenance decisions within the rich context of a specific project.

III. Research Directions

We begin with the assumption that we can perform a full clone detection across a source code base of a given system. Typically, this produces a very large result set which must be investigated systematically in some way, if it is to be of any value. The key question is: How can we best take advantage of the results of the clone detection to aid the quality assessment (QA) process? We list a few hopeful directions.

1. Software Problem-Oriented Taxonomy of Clones

Cloning causes a variety of problems, yet researchers have studied only a few of them, mainly on bugs that are created or propagated by cloning. Cloning is associated with other software problems in many dimensions — for example, increased maintenance complexity, and license infringement [3]. From the perspective of developers, a clone detection report processed by such classifier can greatly enhance its usability compare to a plain detection report. A clone management task is therefore decomposed into several smaller tasks, each of which might be integrated with the existing process to address same underlying software problem. Developers can also target one or a few categories that reflect the requirements of their own projects. By this problem-oriented taxonomy, the task of clone management is incorporated with the existing software management process. The key of this direction, therefore, is to be able to organize the clone detection set according to various perceived problems.

2. Crosschecking With Other Software Artifacts

If we want to build a problem-oriented taxonomy of clones, we may want to crosscheck the clone detection results with other software artifacts — such as the bug repository or design documents — to find "real offenders" of software problems in clones. For example, developers can study the result of crosschecking the clone detection report and the bug repository, to assess bug-related problems in cloned code chunks. The primary goal of this direction is to establish linkage between source code clones and many kinds of software artifacts, by mining or other techniques.

3. Restructuring Clone Detection Result Sets

The sensitivity of a cloned detection result set as an indicator for software defects is no higher than that of non-cloned code [6]. Our experience also suggests that cloned code may not often suggest other kinds of software problems. This low sensitivity issue contributes directly to the cost-effectiveness problem, that is, developers may feel frustrated if investigating large amounts of clones yields little benefits. We may restructure the clones with certain criteria — such as ranking — to find out our "top offenders". The key here is to be able to identify a set of "interestingness" criteria that permits ranking of candidate clones. Such criteria might include size of clone (in LOC or tokens), number of clones in the clone class, clones in code that has changed often, clones in code associated with "bugginess", clones clones across code in different parts of the system (architectural spread), etc.

IV. Evaluation

If we focus only on bugs that are caused or propagated by cloning, then the number of bugs being detected or fixed is the primary evaluation metric of a clone management framework. For a future clone management tool that is designed to cover a wider range of software issues, we need new metrics, especially to measure the improvement of clone maintainability, and perhaps a metric for evaluating the overall effectiveness of a clone management tool.

V. Conclusion

Managing clones is no longer considered a "hunt-and-kill" game. Current clone management techniques focus on providing more detailed contextual information about clones within development environments. This approach allows for flexible management of clones by developers, but also the leaves the burden of decision making — including studying and determining the reason of cloning, and the problems each clone is associated with — entirely to developers.

To bridge the gap from cloning analysis to clone management, we need to answer research questions in many dimensions. What are the common needs of clone management from industry? What are the biggest hindrances to delivering software management solutions from cloning analysis? How can we evaluate the effectiveness of a clone management framework? We predict this line of research would lead to a series fruitful clone management techniques, and ultimately shape the clone management practice in the industry.

References

[1] James R. Cordy. Comprehending reality – practical barriers to industrial adoption of software maintenance automation. *International Conference on Program Comprehension*, 2003.

[2] Ekwa Duala-Ekoko and Martin P. Robillard. Tracking code clones in evolving software. In *Proceedings of the 29th international conference on Software Engineering*, ICSE '07, pages 158–167, Washington, DC, USA, 2007. IEEE Computer Society.

[3] Daniel M. German, Yuki Manabe, and Katsuro Inoue. A sentence-matching method for automatic license identification of source code files. In *Proceedings of the IEEE/ACM international conference on Automated software engineering*, ASE '10, pages 437–446, New York, NY, USA, 2010. ACM.

[4] Jan Harder and Nils Göde. Quo vadis, clone management? In *Proceedings of the 4th International Workshop on Software Clones*, IWSC '10, pages 85–86, New York, NY, USA, 2010. ACM.

[5] Cory Kapser and Michael W. Godfrey. "cloning considered harmful" considered harmful. In *Proceedings of the 13th Working Conference on Reverse Engineering*, pages 19–28, Washington, DC, USA, 2006. IEEE Computer Society.

[6] Foyzur Rahman, Christian Bird, and Premkumar Devanbu. Clones: What *is* that Smell? In *Proceedings of the Seventh Working Conference on Mining Software Repositories*. IEEE Computer Society, 2010.