# Cloning: The Need to Understand Developer Intent

Debarshi Chatterji, Jeffrey C. Carver and Nicholas A. Kraft
University of Alabama, Tuscaloosa, AL, USA
dchatterji@ua.edu carver, nkraft@cs.ua.edu

***Abstract*—Many researchers have studied the positive and negative effects of code clones on software quality. However, little is known about the intent and rationale of the developers who clone code. Studies have shown that reusing code is a common practice for developers while programming, but there are many possible motivations for and approaches to code reuse. Although we have some ideas about the intentions of developers when cloning code, comprehensive research is needed to gather conclusive evidence about these intentions and categorize clones based on them. In this paper we argue that to provide developers with better clone management tools, we need to interview developers to better understand their intentions when managing cloned code.**

***Index Terms*—Code clones, software clones, empirical studies, clone management, software maintenance, developer behavior.**

## I. Introduction

Studies indicate that software maintenance costs can account for up to 90% of the total software lifecycle costs [2], [4]. Some portion of these maintenance costs are the result of using programming practices like code cloning in inappropriate situations. Code cloning is commonly used by programmers to save time and effort over writing new code from scratch [7]. Cloning saves effort by reusing complex frameworks and by replicating design patterns and other high level code structures. Initially, the term code clone had a negative connotation because of the belief that cloned code required additional effort during maintenance (e.g., propagation of defects through cloning code) [8]. More recently, a number of studies indicate that code clones are not harmful to software quality [5], [6], [8], [11]. As this discussion indicates, much of the research about code clones has focused on understanding and evaluating their effects during maintenance. By contrast, little research has focused on understanding developers' rationale for cloning code during development.

In addition, the tools that support automated clone detection have largely been based on the structural definitions of clone types as summarized by Roy et al. [10]. While there is usefulness in this type of categorization, the primary drawback is that it is difficult to use the categorization to evaluate which clones need to be actively managed and which do not require attention. Our previous survey of the clone research community indicated that a large percentage of the survey respondents believed that most clone tools are more useful for educational purposes than for providing assistance to developers performing maintenance tasks. In fact, the consensus was that the tools currently available provide very little or no assistance to developers working on maintenance tasks [3]. Based upon these results, we can argue that the current code clone categorizations do not correspond to the practice of code cloning during development. We can also argue that the types of clones are not obvious to developers and are not helpful in real maintenance tasks.

The previous discussion suggests that most of the code clone research and existing tools have focused on post hoc analysis of clones (i.e., in existing code). While this information can be useful for some tasks, there is also a need to understand the behavior of developers when creating clones. For help during typical development and maintenance scenarios, developers need tools that are designed to address actual use cases.

We argue that one important method for gathering this information is through developer interviews focused on understanding their reasons for cloning code. After identifying these reasons, we can then develop a different type of clone categorization, i.e., one that focuses on developer intent rather than focusing on code structure. Kapser et al. [6] developed a similar categorization regarding developer intent. However, in this study, developer intent was inferred from a post hoc analysis of large software systems. Zhang et al. [12] used interviews to study developer intent. Yet, while they categorized the developer intentions, they did not use their findings to categorize code clones.

## II. Cloning Intent and Rationale

We argue that whether cloning is likely to have a positive effect or a negative effect can, at least partially, be traced to the intention of developers. We also argue that intentional clones may have a different effect than unintentional clones.

First, regarding intentional clones, some types of cloning that are likely to have a negative effect are: duplicating code, hard coding rather than applying a proper abstraction, and reusing code because of limited understanding of the code fragment. These cloning practices can give rise to unnecessary coupling and propagate bugs throughout the system. Conversely, in some cases intentional cloning can prove to be beneficial. Intentional cloning of robust code architectures with desired behavior, e.g., design patterns, can improve the readability and the overall quality of the system. Another positive example of intentional cloning is re-using an older, tested version of the software or a platform equivalent as a 'springboard' for the software development. A significant amount of time and effort can be saved over rebuilding the same architecture from scratch.

Second, unintentional clones may pose a greater threat to the quality of the system. While unintentional clones do not always have a negative impact, they can be easily overlooked due to their unintentional nature. An example of the source of an unintentional clone is that developers often solve a specific type of problem in a similar way each time. While the code may appear to be cloned, it may simply be that developers have programming idioms and recurring constructs that they frequently use [1]. In this case, a static, post hoc analysis of the code would indicate the presence of a clone, while an understanding of developer intent would indicate otherwise.

## III. Categorization Based on Intent

In Kapser et al.'s categorization of clones based on intent [6], they described 11 high-level patterns of code cloning divided into four categories: *Forking*, *Templating*, *Customization* and *Exact matches*. Brief descriptions of these follow

- *Forking*: large software artifacts are used as "springboards" for new development. The newer versions are expected to evolve independently from the original code.
- *Templating*: a known solution to a problem is reused and modified. For example, templating can bue used when it makes more sense to reuse code with modification than to abstract the similar code into a common base.
- *Customization*: reusing a known solution with extensions or modifications to fit the current situation.
- *Exact matches*: repeatedly using a clone fragment throughout a system because either the fragment is too insignificant to justify abstraction or it cannot be used outside of its original context.

While we believe this categorization may be useful, it was inferred only from a post hoc analysis of the code from large software systems. There is a need to validate whether these intentions are really how developers would categorize their behavior.

Zhang et al. [12] studied the intent behind cloning practices by interviewing developers. In their paper they categorize cloning intentions based on technical, personal and organizational reasons. However, the do not provide a categorization of the clones based on the reasons for cloning that they found in their study. We propose to conduct developer interviews as a method for understanding intentions and for using that understanding to categorize code clones.

## IV. Developer Interviews

Our motivation for using developer interviews to gather this information is motivated by two primary objectives. First, this type of study will provide a solid foundation for categorization based on cloning intent and use cases of management. Second, a categorization based on cloning intent can induce further research on clone management tools. The tools can be tailor made for assisting developers in specific maintenance scenarios. To achieve these objectives, developers should be interviewed to understand why they clone code fragments or code structures. However, further validation and conclusive evidence is required.

Retrospective analysis of source code, like in Kapser et al.'s study [6], can reveal 'what' developers did. However, the aim should be to evaluate the reasons 'why' developers clone code and further classify them into good or bad practices. Roehm et al.'s study showed that observing code can reveal what developers do. However, to understand their intentions behind their actions it is important to directly interview them [9].

The intent part of the information behind a maintenance task is equally crucial and helps to validate the observations from post hoc analysis of the system. For example, Thummalapenta et al. [11] inferred from the results of their source code analysis that developers of four subject systems knew when and how to propagate clone changes, but understanding the developers' intent is critical to validating this inference.

## V. Conclusion

To understand the needs of real life clone management we need to understand why developers take specific actions that give rise to clones. In this paper we focused on the requirement of interviewing developers. Although observing code is important for analyzing patterns, understanding the intent and rationale behind actions that give rise to such patterns is of equal importance. Human behavior has various confounding factors and to judge the effects correctly the only way is to reach out to the real life developers who handle such situations in practice.

## Acknowledgment

We acknowledge support from NSF grant CCF-0915559.

## References

[1] I. Baxter, A. Yahin, L. Moura, M. Sant'Anna, and L. Bier, "Clone detection using abstract syntax trees," in *ProcInt'l Conf. on Software Maintenance*, 1998, pp. 368 –377.

[2] B. Boehm and V. R. Basili, "Software defect reduction top 10 list," *Computer*, vol. 34, no. 1, pp. 135–137, Jan. 2001.

[3] D. Chatterji, J. C. Carver, and N. A. Kraft, "Clone research community beliefs about code clones and developer behavior — two surveys," Department of Computer Science, The University of Alabama, Tech. Rep. SERG-2013-01, 2013. [Online]. Available: http://software.eng.ua.edu/reports/SERG-2013-01

[4] L. Erlikh, "Leveraging legacy system dollars for e-business," *IT Professional*, vol. 2, no. 3, pp. 17–23, May/June 2000.

[5] R. Geiger, B. Fluri, H. C. Gall, and M. Pinzger, "Relation of code clones and change couplings," in *Proc. Int'l Conf. on Fundamental Approaches to Software Engineering*, 2006, pp. 411–425.

[6] C. J. Kapser and M. W. Godfrey, ""cloning considered harmful" considered harmful: patterns of cloning in software," *Empirical Software Engineering*, vol. 13, no. 6, pp. 645–692, Dec. 2008.

[7] M. Kim, L. Bergman, T. Lau, and D. Notkin, "An ethnographic study of copy and paste programming practices in oopl," in *Proc. Int'l Sym. on Empirical Software Engineering*, 2004, pp. 83–92.

[8] F. Rahman, C. Bird, and P. Devanbu, "Clones: What is that smell?" in *Proc. IEEE Working Conf. on Mining Software Repositories*, 2010, pp. 72–81.

[9] T. Roehm, R. Tiarks, R. Koschke, and W. Maalej, "How do professional developers comprehend software?" in *Proc. Int'l Conf. on Software Engineering*, 2012, pp. 255 –265.

[10] C. K. Roy, J. R. Cordy, and R. Koschke, "Comparison and evaluation of code clone detection techniques and tools: A qualitative approach," *Sci. Comput. Program.*, vol. 74, no. 7, pp. 470–495, May 2009.

[11] S. Thummalapenta, L. Cerulo, L. Aversano, and M. Di Penta, "An empirical study on the maintenance of source code clones," *Empirical Software Engineering*, vol. 15, no. 1, pp. 1–34, Feb. 2010.

[12] G. Zhang, X. Peng, Z. Xing, and W. Zhao, "Cloning practices: Why developers clone and what can be changed," in *Proc. IEEE Int'l Conf. Software Maintenance*, 2012, pp. 285–294.