2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)

# Reducing accidental clones using instant clone search in automatic code review

Vipin Balachandran
VMware, Palo Alto, USA
vbala@vmware.com

***Abstract*—Accidental clones occur when developers are not familiar with the codebase. We propose changes in the developer code review workflow to leverage online clone detection to identify duplicate code during code review. A developer survey's responses indicate that the proposed workflow change will increase the usage of clone detection tools and can reduce accidental clones.**

## I. Introduction

Code duplication increases software maintenance cost [1]–[3], leads to incorrect program behavior due to inconsistent changes to cloned fragments [4], and affects software quality [3], [4]. According to Juergens et al. [4], a significant portion of non-identical clones is created unintentionally. This type of cloning is referred to as *cloning by accident* [3], [5]. The accidental clones are created when the developers are not familiar with the entire codebase. The developers working on different modules of a software project may re-implement methods if they are not aware of existing implementations. Kononenko et al. [6] report that the biggest challenge for developers during code reviews is gaining familiarity with the codebase [6].

It is observed that the majority of developers in our organization do not use clone detection tools before merging the code. This observation is consistent with the findings of Johnson et al. [7] concerning the usage of static analysis tools. Integrating clone detection tools with the build process is not sufficient because the duplicate code is often merged by the time the build is generated. It is important to show program analysis reports early in the development (preferably before merging the code) when developers are more likely to address any problem [8], [9]. As mentioned in [8], build integration is not practical when the analysis is time-consuming, or required to restrict the analysis to changes in a diff. Also, the build reports do not facilitate developer discussions that are essential before attempting non-trivial refactoring to remove the duplicate code.

Similar to the observations by Sadowski et al. [8], we also observed that the usage of program analysis tools is low when developers have to context switch to run the tools. Integration into standard developer workflows is required for the effective usage of program analysis tools [7]. Peer code review is the most important process for reducing defects and improving software quality [6]. The light-weight, tool-based code review, also known as modern code review [10], is widely adopted by the software industry [10]–[12] and open-source projects [6], [13]. The TRICORDER program analysis platform from Google [8] and the Review Bot from VMware [11] have shown that the integration of static analysis tools with modern code review improved software quality. Inspired by these findings, we propose integrating clone detection into tool-based code reviews to reduce accidental clones.

The program analysis report shown at code review time ensures peer accountability [8]. Therefore, automatic code review generated from the clone detection tool's report is an effective way to ensure that such reports are analyzed. It eliminates context switches and enables discussions about refactoring using the threaded conversations supported by modern code review tools. The review comments should be restricted to the changed lines to keep the code review relevant to the diff under consideration. Therefore, the clone detection tool should be able to find duplicates similar to a code snippet. Also, it should scale to large repositories since developers are less likely to act on delayed analysis reports [8]. These requirements necessitate the use of real-time [14] or instant clone search tools [15] to generate an automatic review.

The changes in code review workflow will not be successful without developer acceptance. To gauge the interest and guide the tool's design, we conducted a developer survey. Section II provides an overview of our proposal and discusses the survey results. The related work is discussed in Section III, and we conclude in Section IV.

## II. Workflow Integration

The code snippet search discussed in [16] called *Query by Example* (QBE) satisfies our instant clone search requirements because it is possible to query smaller code fragments within a few seconds. Also, it scales well to large projects. We implemented an automatic code review tool named RBOT-CC for Java projects hosted in GitLab [17] repositories. When notified of a pull request, RBOT-CC downloads and applies the diff files to generate the patched source files. The ASTs are generated for each of the source files. It then queries the QBE server for the source corresponding to the methods (identified by AST node type) modified by the diff files. Finally, the query result is used to generate the review comments.

### A. Developer Survey

We prepared a survey consisting of 7 multiple-choice questions to answer the following research questions:

- **RQ$_1$**: Does the proposed workflow integration increase the usage of clone detection tools?
- **RQ$_2$**: What information developers expect in code review comments that indicate potential duplication?
- **RQ$_3$**: What factors affect the developer's decision to address or reject such review comments?

The survey had a short description of accidental clones before presenting the questions. We sent the survey to 30 developers working in Cisco, Google, Microsoft, and VMware, where tool-based modern code review workflows are standard practice [8], [10]–[12], [18]. The survey was open for one week, and we received 17 responses (57% response rate). Even though the number of survey responses is small, it is comparable to a similar study [7]. Also, the goals of this study are feasibility analysis and to provide guidelines for the tool design and not to evaluate the effectiveness of the workflow modification. The respondents' industry experience ranges from 2 to 21 years, with a median of 11 years. Since it was optional to reveal the identity, the majority of the responses were anonymous. Therefore, we could not correlate responses to specific companies.

To answer RQ$_1$, we asked the following questions:

- $Q_1$: How often do you run a code clone detection tool before merging the code?
- $Q_2$: How do you react to a code review comment which indicates that you might have duplicated a method unintentionally?

Nine out of 17 developers responded that they do not run any tool, while five developers responded they rarely do. Only three developers responded that they run a tool most of the time. This response is consistent with the usage of static analysis tools among developers [7]. For $Q_2$, all 17 developers responded that they would look into the issue described in the comment. This higher acceptance rate may be partly due to the peer accountability enabled by code reviews. Based on these responses, we can answer RQ$_1$– integration of code clone detection tool into modern code review can increase its usage.

We asked the developers the information they look for in a review comment ($Q_3$) that notifies code duplication to study RQ$_2$. Along with the pre-defined choices, we also provided a field for a free-text response. Fourteen developers responded that they are interested in the path and the signature of the original code, and ten developers said that they would like to see the original code snippet. We will use these responses to decide the review comment format of RBOT-CC.

We used the following question to study RQ$_3$:

- $Q_4$: What factors influence your decision to address the review comment (i.e., remove the duplicate code)?

There were pre-defined choices and an option for a free-text response. We also asked two related questions to understand the specific cases that influence the acceptance ($Q_5$) or the rejection ($Q_6$) of the review comment. The main factor that influences the decision is the complexity of the refactoring involved (10 developers). The other factors are: (i) the criticality of the original method that needs refactoring (8 responses); (ii) the consensus reached based on the discussion with code reviewers (7 responses); and (iii) the urgency of the current patch (5 responses). The developers indicated that they would address the review comment if the refactoring is localized to the new code, or the duplicate code is an identical clone (11 responses each). Seven developers mentioned that they would address the comment if the original code contains a bug that is fixed in the current patch. Following are the main reasons for rejecting a comment: (i) the refactoring is complex and affects files/services owned by others (13 responses); (ii) the original method is in several critical paths and hence the refactoring is risky (11 responses); and (iii) the current change is fixing a high priority bug and hence should be merged quickly (9 responses). As a follow-up question to $Q_6$, we asked the developers whether they would like to create a task to track the refactoring if it cannot be addressed in the current change ($Q_7$), and 15 out of 17 developers responded positively. Two developers responded that their decision would depend on the duplicate code.

## III. Related Work

Review Bot [11] is a code review tool that uses Checkstyle [19], PMD [20], and FindBugs [21] to generate automatic code reviews to detect coding standard violations and bugs. In [8], the authors proposed a program analysis platform that integrates static analysis tools into code review. Upsource [22] is a code review tool that generates automatic reviews using static analysis tools. IntelliJ IDEA [23] is an IDE that has minimal support for detecting repetitive blocks of code due to copy-paste. To the best of our knowledge, this is the first paper that discusses the integration of code clone detection into code review workflow.

## IV. Conclusions and Future Work

In this paper, we proposed using online clone detection for automatic code review generation to detect and remove accidental clones. Through a developer study, we have shown that the proposed workflow integration will improve the usage of code clone detection tools and therefore reduce accidental clones. The survey results show that the developers rarely run stand-alone clone detection tools. The research community should focus more on online clone detection tools, which can be better integrated into existing developer workflows.

We are planning to incorporate the insights learned from the survey responses in the comment generation of RBOT-CC. The developers do not prefer to refactor the original code if it is critical, but their perception of criticality may not be accurate. To enable the developers to make an informed decision, we consider incorporating the NodeRank [24] value of the method that measures its relative importance in the review comment. Another option is to integrate a source code browser that enables searching all the references of a method. The next step is to qualitatively measure the effectiveness of the proposed workflow integration using a developer study.

## REFERENCES


[1] A. Monden, D. Nakae, T. Kamiya, S. Sato, and K. Matsumoto, "Software quality analysis by code clones in industrial legacy software," in *Proceedings Eighth IEEE Symposium on Software Metrics*, 2002, pp. 87–94.

[2] R. Koschke, "Survey of research on software clones," in *Dagstuhl Seminar Proceedings*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2007.

[3] C. K. Roy and J. R. Cordy, "A survey on software clone detection research," *Queen's School of Computing TR*, vol. 541, no. 115, pp. 64–68, 2007.

[4] E. Juergens, F. Deissenboeck, B. Hummel, and S. Wagner, "Do code clones matter?" in *2009 IEEE 31st International Conference on Software Engineering*, 2009, pp. 485–495.

[5] R. Al-Ekram, C. Kapser, R. Holt, and M. Godfrey, "Cloning by accident: an empirical study of source code cloning across software systems," in *2005 International Symposium on Empirical Software Engineering, 2005.*, 2005, pp. 10 pp.–.

[6] O. Kononenko, O. Baysal, and M. W. Godfrey, "Code review quality: How developers see it," in *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, 2016, pp. 1028–1038.

[7] B. Johnson, Y. Song, E. Murphy-Hill, and R. Bowdidge, "Why don't software developers use static analysis tools to find bugs?" in *2013 35th International Conference on Software Engineering (ICSE)*, 2013, pp. 672–681.

[8] C. Sadowski, J. v. Gogh, C. Jaspan, E. Söderberg, and C. Winter, "Tricorder: Building a program analysis ecosystem," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol. 1, 2015, pp. 598–608.

[9] N. Ayewah, W. Pugh, D. Hovemeyer, J. D. Morgenthaler, and J. Penix, "Using static analysis to find bugs," *IEEE software*, vol. 25, no. 5, pp. 22–29, 2008.

[10] A. Bacchelli and C. Bird, "Expectations, outcomes, and challenges of modern code review," in *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, 2013, pp. 712–721.

[11] V. Balachandran, "Reducing human effort and improving quality in peer code reviews using automatic static analysis and reviewer recommendation," in *2013 35th International Conference on Software Engineering (ICSE)*, 2013, pp. 931–940.

[12] C. Sadowski, E. Söderberg, L. Church, M. Sipko, and A. Bacchelli, "Modern code review: a case study at google," in *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice*, 2018, pp. 181–190.

[13] F. E. Zanaty, T. Hirao, S. McIntosh, A. Ihara, and K. Matsumoto, "An empirical study of design discussions in code review," in *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, 2018, pp. 1–10.

[14] I. Keivanloo, J. Rilling, and P. Charland, "Internet-scale real-time code clone search via multi-level indexing," in *2011 18th Working Conference on Reverse Engineering*. IEEE, 2011, pp. 23–27.

[15] M.-W. Lee, J.-W. Roh, S.-w. Hwang, and S. Kim, "Instant code clone search," in *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering*, 2010, pp. 167–176.

[16] V. Balachandran, "Query by example in large-scale code repositories," in *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2015, pp. 467–476.

[17] "GitLab," https://about.gitlab.com/.

[18] J. Cohen, "Best kept secrets of peer code review. smart bear," *Inc., Austin, TX*, p. 117, 2006.

[19] "Checkstyle," https://checkstyle.sourceforge.io/.

[20] "PMD Source Code Analyzer," https://pmd.github.io/.

[21] "FindBugs," http://findbugs.sourceforge.net/.

[22] "Upsource," https://www.jetbrains.com/upsource/.

[23] "IntelliJ IDEA," https://www.jetbrains.com/idea/.

[24] P. Bhattacharya, M. Iliofotou, I. Neamtiu, and M. Faloutsos, "Graph-based analysis and prediction for software evolution," in *2012 34th International Conference on Software Engineering (ICSE)*, 2012, pp. 419–429.