# PRECINCT: An Incremental Algorithm to Prevent Clone Insertion

Mathieu Nayrolles,
Software Behaviour Analysis (SBA) Research Lab
ECE, Concordia University
Montreal, Canada
m_nayrol@ece.concordia.ca

Abdelwahab Hamou-Lhadj
Software Behaviour Analysis (SBA) Research Lab
ECE, Concordia University
Montreal, Canada
abdelw@ece.concordia.ca

***Abstract*—Software clones are considered harmful in software maintenance and evolution. However, after a decade and a half of research, only few approaches have targeted the prevention aspect of clones detection. In this paper, we propose a novel approach named PRECINCT (PREventing Clones INsertion at Commit Time). PRECINCT focuses on near-miss software clones are copied — or reinvented — fragments where minor to extensive modifications have been made and more specifically on detecting at commit time by means of pre-commit hooks. Efficiently detecting near-miss software clones at commit time might call for further refactoring or simply hint developers that they reinvented one piece of code. We apply and validate PRECINCT in terms of precision and recall on seven systems developed independently with a wide range of technologies, size and purposes. The validation demonstrates that our approach detects near-miss software clones before they reach the source version system with a 100% precision and a 93% recall.**

## I. Introduction

Code or software clones appear when developers reuse code with little to no modification. Previous research has shown that clones can account for 7% to 50% of a given software [1], [2]. Developers often reuse code and create clones in their software on purpose [3]. Nevertheless, clones are considered a bad practice in software development and therefore, harmful [4]–[6]. The most obvious way a clone can hazardous for the quality of a software system is if a default or bug is discovered in one segment of code that has been copied pasted several times, then the developers would have to remember the places where this segment has been reused in order to fix the default in each of them.

In order to help developers to deal with clones, researchers and practitioners have published hundreds of studies and dozens of tools using different approaches: (1) textual where the source code is considered as text and transformation or normalization is applied to it in order to compare it with order code fragment [7]–[10]. (2) Lexical where the source code is sliced into sequences of tokens as a compiler would [1], [6], [11]–[13]. (3) Syntactic where the source code is converted into trees, more particularly abstract syntax tree (AST) and then, the clone detection is performed using tree matching algorithms [14]–[17].

Henceforth, clones detection can be considered as a mature and still active field of research with two decades of research and hundreds of publications. Consequently, practitioners and engineers know that copy-pasting segment of code can hinder the maintenance and the quality of their systems and that approaches exist to detect them. However, the use of such approaches and tools is not as widespread as one might think. Indeed, as for automatic bug finding tools, the main reasons for this lack of infatuation are that these tools are known to have (1) massive outputs which is (2) hard to understand and contain (3) a high amount of false positives. Moreover, these tools are (4) hard to configure and there is (5) little to no integration of these tools in the day-to-day workflow of a developer [18].

In this paper, we present PRECINCT (PREventing Clones INsertion at Commit Time) that focuses on preventing the insertion of clones at commit time. More specifically, our approach provides a clone detection process that eases the five major reasons that limit the adoption of such tools. To do so, we use pre-commit hooks capabilities of modern source code version control. A pre-commit hook is a process that one can implements to receive the latest modification to the source code done by a given developer just before the code reaches the central repository. Then, analysis can be done using bash programming or calling external programs, allowing the changes to go through or not. PRECINCT is, in fact, a pre-commit hook that detects clones that might have been inserted in the latest changes with regard to the rest of the source code. Consequently, only a fraction of the code is analyzed and PRECINCTS reduce the output by 70% (compared to the clone detector PRECINCTS is built upon NiCad [9]). Moreover, the detected clones are presented in using a classical diff output that developers are used to. Hence, concerns (1) *massive outputs* and (2) *hard to understand* are less than in other tools. Also, PRECINCTS can be installed and configured using only one command line (*hard to configure*). Finally, our approach leverages the pre-commit hook capabilities of modern source code version control and therefore, integrates itself at the heart of the programmers workflow (*little to no integration of these tools in the day-to-day workflow*).

We assessed the capabilities of PRECINCT in terms of precision and recall on seven systems developed independently with a wide range of technologies and purposes. The validation demonstrates that our approach detects near-miss software clones before they reach the source version system with a

100% precision and a 93% recall while increasing the size of the repository by only 3% to 5%.

The rest of this paper is organized as follows: In Section II we present the works related to PRECINCT and with a particular attention of Nicad. Then, in Section III we present the PRECINCT approach and Section IV shows the experimentations we conduct to assess the capacities of PRECINCT. Finally, we propose some concluding remarks in Section V.

## II. Related Works

## III. The PRECINCT Approach

## IV. Experimentations

## V. Conclusion

## References


[1] B. Baker, "On finding duplication and near-duplication in large software systems," in *Proceedings of 2nd Working Conference on Reverse Engineering*. IEEE Comput. Soc. Press, pp. 86–95. [Online]. Available: http://ieeexplore.ieee.org/articleDetails.jsp?arnumber=514697

[2] S. D. Stéphane Ducasse, Matthias Rieger, "A Language Independent Approach for Detecting Duplicated Code." [Online]. Available: http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.31.6060

[3] M. Kim, V. Sazawal, and D. Notkin, "An empirical study of code clone genealogies," *ACM SIGSOFT Software Engineering Notes*, vol. 30, no. 5, p. 187, sep 2005. [Online]. Available: http://dl.acm.org/citation.cfm?id=1095430.1081737

[4] C. Kapser and M. Godfrey, ""Cloning Considered Harmful" Considered Harmful," in *2006 13th Working Conference on Reverse Engineering*. IEEE, oct 2006, pp. 19–28. [Online]. Available: http://ieeexplore.ieee.org/articleDetails.jsp?arnumber=4023973

[5] E. Juergens, F. Deissenboeck, B. Hummel, and S. Wagner, "Do code clones matter?" in *2009 IEEE 31st International Conference on Software Engineering*. IEEE, may 2009, pp. 485–495. [Online]. Available: http://dl.acm.org/citation.cfm?id=1555001.1555062

[6] Z. Li, S. Lu, S. Myagmar, and Y. Zhou, "CP-Miner: finding copy-paste and related bugs in large-scale software code," *IEEE Transactions on Software Engineering*, vol. 32, no. 3, pp. 176–192, mar 2006. [Online]. Available: http://ieeexplore.ieee.org/articleDetails.jsp?arnumber=1610609

[7] J. H. Johnson, "Visualizing textual redundancy in legacy source," p. 32, oct 1994. [Online]. Available: http://dl.acm.org/citation.cfm?id=782185.782217

[8] ——, "Identifying redundancy in source code using fingerprints," pp. 171–183, oct 1993. [Online]. Available: http://dl.acm.org/citation.cfm?id=962289.962305

[9] J. R. Cordy and C. K. Roy, "The NiCad Clone Detector," in *2011 IEEE 19th International Conference on Program Comprehension*. IEEE, jun 2011, pp. 219–220. [Online]. Available: http://ieeexplore.ieee.org/articleDetails.jsp?arnumber=5970189

[10] C. Roy and J. Cordy, "NICAD: Accurate Detection of Near-Miss Intentional Clones Using Flexible Pretty-Printing and Code Normalization," in *2008 16th IEEE International Conference on Program Comprehension*. IEEE, jun 2008, pp. 172–181. [Online]. Available: http://ieeexplore.ieee.org/articleDetails.jsp?arnumber=4556129

[11] B. S. Baker, "A program for identifying duplicated code." [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.550.4540

[12] B. S. Baker and R. Giancarlo, "Sparse Dynamic Programming for Longest Common Subsequence from Fragments," *Journal of Algorithms*, vol. 42, no. 2, pp. 231–254, feb 2002. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0196677402912149

[13] T. Kamiya, S. Kusumoto, and K. Inoue, "CCFinder: a multilinguistic token-based code clone detection system for large scale source code," *IEEE Transactions on Software Engineering*, vol. 28, no. 7, pp. 654–670, jul 2002. [Online]. Available: http://dl.acm.org/citation.cfm?id=636188.636191

[14] I. D. Baxter, A. Yahin, L. Moura, M. Sant'Anna, and L. Bier, "Clone Detection Using Abstract Syntax Trees," p. 368, mar 1998. [Online]. Available: http://dl.acm.org/citation.cfm?id=850947.853341

[15] R. Komondoor and S. Horwitz, "Semantics-preserving procedure extraction," in *Proceedings of the 27th ACM SIGPLAN-SIGACT symposium on Principles of programming languages - POPL '00*. New York, New York, USA: ACM Press, jan 2000, pp. 155–169. [Online]. Available: http://dl.acm.org/citation.cfm?id=325694.325713

[16] R. Tairas and J. Gray, "Phoenix-based clone detection using suffix trees," in *Proceedings of the 44th annual southeast regional conference on - ACM-SE 44*. New York, New York, USA: ACM Press, mar 2006, p. 679. [Online]. Available: http://dl.acm.org/citation.cfm?id=1185448.1185597

[17] R. Falke, P. Frenzel, and R. Koschke, "Empirical evaluation of clone detection using syntax suffix trees," *Empirical Software Engineering*, vol. 13, no. 6, pp. 601–643, jul 2008. [Online]. Available: http://link.springer.com/10.1007/s10664-008-9073-9

[18] B. Johnson, Y. Song, E. Murphy-Hill, and R. Bowdidge, "Why don't software developers use static analysis tools to find bugs?" pp. 672–681, may 2013. [Online]. Available: http://dl.acm.org/citation.cfm?id=2486788.2486877

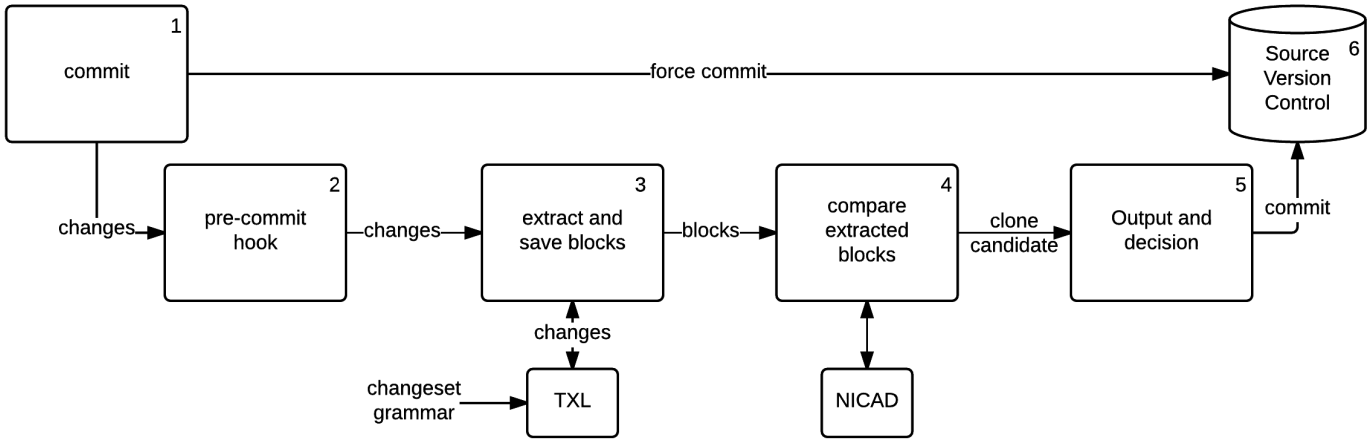


Fig. 1. Overview of the PRECINCT Approach.