Hot Clones: Combining Search-Driven Development, Clone Management, and Code Provenance

Niko Schwarz
University of Bern

***Abstract*—Code duplication is common in current programming-practice: programmers search for snippets of code, incorporate them into their projects and then modify them to their needs. In today's practice, no automated scheme is in place to inform both parties of any distant changes of the code. As code snippets continue to evolve both on the side of the user and on the side of the author, both may wish to benefit from remote bug fixes or refinements — authors may be interested in the actual usage of their code snippets, and researchers could gather information on clone usage. We propose to maintain a link between software clones across repositories and outline how the links can be created and maintained.**

***Keywords*-clone detection; software maintenance; corrective clone management**

I. Introduction

Since the rise of internet-scale code search engines, searching for reusable source code has become a fundamental activity for developers [1]. Developers use search engines to find and reuse software. The promise of search-driven development is that developers will save time and resources by using search results. However, there are perils: the current practice of manually integrating code search results into a local code base leads to a proliferation of untracked code clones. As a side effect, in the authors' experience, bugs fixed in one clone typically do not traverse their new environment anymore, and the same holds true for extensions and code cleanups.

Even if they appear in the same project, software clones often cannot be eliminated [2]. But oversights in consistently applying changes to clones may introduce bugs into the system [3]. Therefore, tools have been proposed to maintain links between software clones [4], [3], [5], but they fail to link clones that are beyond project boundaries. Codebook by Begel *et al.* [6], is a social network in which people can befriend both other people and the artifacts they produce. Codebook is intended to maintain links between clones, it is however unclear how these links come into being. Begel *et al.* only vaguely suggest how that should be done: edges are to be added between a definition and its likely clones.

In this paper, we propose a scheme to initially create and then maintain such links. A code search engine assists the developer by integrating its results into the source code. The IDE then remembers the origin of the code snippet and informs the repository that a clone was created, thus creating a link between original and copy. We will refer to clones created in this way as hot clones. Whenever a hot clone changes, the linked clones' developers are informed and offered the option to update their instance. Also, whenever a method is inspected, its clones can be inspected too, providing valuable information.

In the terminology of Koschke [7], we provide *compensative clone management*, *i.e.,* we limit the negative impact of existing clones, but we also give developers benefits from the software duplication introduced by clones by providing developers with information on how their code is used and modified.

This paper is an updated version of previous work [8]. It differs in that Section §III outlines the wide applicability of a *hot clones* search engine, but does not discuss the integration into the development process.

II. Goals

The idea of hot clones scratches two itches. The first is to let developers benefit from code cloning, and the second is to provide researchers with more information on how cloning is used.

We believe that hot clones can ease backporting changes that occur in a linked clone. We believe that during development, hot clones will provide important feedback to developers. Contrasting one's own code with modified clones will give hints to bugs in related code, usage patterns, and plain examples of usage.

The nature of clones within a single software project has previously been studied [9], but the evolution of code snippets copied from searches in software projects has not. A prototypical implementation of hot clones would provide this opportunity. Being able to track the further evolution of code snippets after they are copied out of a search engine may give us great insight into the evolution of code, beyond the classification provided by Kapser and Godfrey [9]. If used by only a few developers, hot clones can provide insights from both a larger set of data than before, and from a wider range of uses. We plan to provide a prototype of hot clones.

III. Scope

In this section, we will discuss which tasks hot clones can assume and how this suits our goals. Software clones are not typically exact copies of each other, but rather they start as exact copies and then evolve in different ways, for a

978-1-4673-1067-3/12/$31.00 © 2012 IEEE

ICSE 2012, Zurich, Switzerland
ACM Student Research Competition

number of reasons in accordance with the specific needs in their respective environments [2], [9].

Our approach [10] follows the following scheme. An artifact is reduced to a signature, and then an entry in a big table is stored, which maps that signature to its origin. This storage scheme scales very well.

Since every artifact has at least one binary representation, the simplest signature is a hash of that representation. To allow for similar but different things to be identified, the signatures are derived from abstractions, *i.e.,* reductions of the original artifact by stripping away minutiae detail [11] that are not important for overall similarity. In previous work [10], we outlined abstractions that map similar code snippets to the same hashes, by discarding three quarters of the vocabulary of a snippet. Such abstractions are in no way restricted to source code. For example, reducing the resolution of a bitmap image is a simple way of abstracting away minutiae that are unimportant for overall similarity (albeit this step alone is probably insufficient).

Since software diverges, it is important to search across all versions of known software projects, in order to detect a *hot clone*. Once the same snippet was found in two different places, it can be traced in both places individually, in order to provide valuable feedback to the authors on both ends [8].

A big database of signatures mapping to sources represents an invaluable source for future research, going even beyond our use case of hot clones. As a rule of thumb, with massive amounts of data to help, difficult problems can suddenly become a lot easier [12]. The following are examples.

- *License compliance.* An organization needs to identify the origin of the software they create (either locally created or licensed) in order to verify that it has satisfied any legal obligations.
- *Security.* The origin of a copy is likely to continue to evolve. It is important to know if any of the copied artifacts contain security related bugs that have been fixed after the copy was made.
- *Verification of binaries.* A customer who subcontracts somebody to develop a software system might have received both source code and binaries. In this scenario, the customer might want to verify that the binaries provided come exactly from the provided source code.
- *Plagiarism.* The owner of the original artifact might want to verify if a copy has been improperly made. In this scenario, the owner might want to find copies of her software artifacts.

IV. Obtained Results

We store all hashes found in the entire software ecosystem in one big table in a relational Postgresql database. We have built up a service that continuously scans the Squeaksource code repository, and updates our repository of hashes of source code.

As a side effect of having built up a large repository of bad hashes on an entire repository, we are able to assess how how many clones are duplicated across different projects for different types of software clones. For type-1, type-2 and type-3 clones, we have estimated the prevalence of software cloning to be between 14 and 18 % [10].

References

[1] S. P. Reiss, "Semantics-based code search," in *ICSE Companion*, vol. 0. Los Alamitos, CA, USA: IEEE Computer Society, 2009, pp. 243–253. [Online]. Available: http://dx.doi.org/10.1109/ICSE.2009.5070525

[2] M. Kim, V. Sazawal, D. Notkin, and G. Murphy, "An empirical study of code clone genealogies," *SIGSOFT Softw. Eng. Notes*, vol. 30, no. 5, pp. 187–196, September 2005. [Online]. Available: http://dx.doi.org/10.1145/1095430.1081737

[3] E. D. Ekoko and M. P. Robillard, "Clonetracker: tool support for code clone management," in *ICSE '08*. New York, NY, USA: ACM, 2008, pp. 843–846. [Online]. Available: http://dx.doi.org/10.1145/1368088.1368218

[4] D. Hou, P. Jablonski, and F. Jacob, "Cnp: Towards an environment for the proactive management of copy-and-paste programming," in *2009 IEEE 17th ICPC*. IEEE, May 2009, pp. 238–242. [Online]. Available: http://dx.doi.org/10.1109/ICPC.2009.5090049

[5] M. Toomim, A. Begel, and S. L. Graham, "Managing duplicated code with linked editing," in *VLHCC '04*. Washington, DC, USA: IEEE, 2004, pp. 173–180. [Online]. Available: http://dx.doi.org/10.1109/VLHCC.2004.35

[6] A. Begel and R. DeLine, "Codebook: Social networking over code," in *ICSE Companion*, 2009, pp. 263–266.

[7] R. Koschke, "Identifying and removing software clones," in *Software Evolution*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, ch. 2, pp. 15–36. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-76440-3_2

[8] N. Schwarz, "Hot clones: a shotgun marriage of search-driven development and clone management," CSMR 2012, doctoral symposium. To Appear., 2012.

[9] C. Kapser and M. W. Godfrey, ""cloning considered harmful" considered harmful," *WCRE '06*, vol. 0, pp. 19–28, 2006. [Online]. Available: http://dx.doi.org/10.1109/WCRE.2006.1

[10] N. Schwarz, M. Lungu, and R. Robbes, "On how often code is cloned across repositories," ICSE NIER 2012. To appear., 2012.

[11] J. Davies, D. M. German, M. W. Godfrey, and A. Hindle, "Software bertillonage: finding the provenance of an entity," in *Proceedings of the 8th Working Conference on Mining Software Repositories*, ser. MSR '11. New York, NY, USA: ACM, 2011, pp. 183–192. [Online]. Available: http://dx.doi.org/10.1145/1985441.1985468

[12] A. Halevy, P. Norvig, and F. Pereira, "The unreasonable effectiveness of data," *IEEE Intelligent Systems*, vol. 24, pp. 8–12, 2009.