Hot Clones: a Shotgun Marriage of Search-Driven Development and Clone Management

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.**

***Index Terms*—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 (*e.g.,* [1], [2]). 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 [3]. But oversights in consistently applying changes to clones may introduce bugs into the system [4]. Therefore, tools have been proposed to maintain links between software clones [5], [4], [6], but they fail to link clones that are beyond project boundaries. Codebook by Begel *et al.* [7], 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. The connections between clone instances are thus proactive and bidirectional.

In the terminology of Koschke [8], 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 [9]. It differs in that it adds Section §V, which outlines our intermediate results.

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 (*e.g.,* [10])), 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 [10]. 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 number of reasons in accordance with the specific needs in their respective environments [3], [10].

To cope with the changes that search-driven development introduces between clone instances, the connections between clone instances should be aware of the semantics of their differences. To retain an active connection between instances

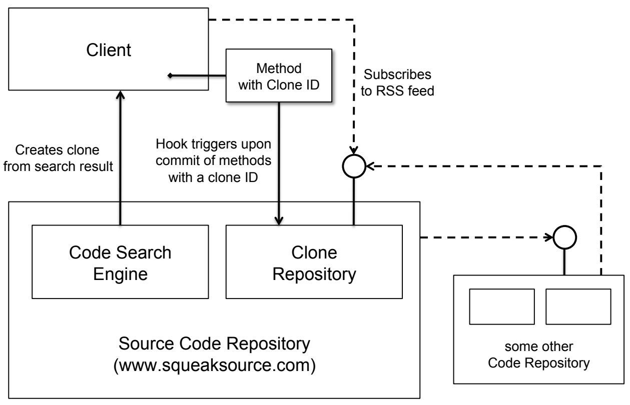

Figure 1 – Proposed architecture of the "hot clone" prototype. It extends MONTICELLO, a distributed version control system for Smalltalk, with two components: a code search engine and a clone repository.

of a hot clone, it is paramount that the clones not only be aware of the lexical but also the semantic changes between their instances [11]. For example, a renamed method call in two different instances may be due to adherence to coding conventions, but may also be due to a deliberate change in one of the instances. So when owners of clones are informed about an incoming change, the semantics of the change should be taken into account. We propose to capture each change in a Changebox [12] annotated with the semantics of a change.

Whenever an incoming change is presented to the developer, the semantic change history of all involved clones has to be taken into account. To consider an example, if the code search engine initially renamed all variable names from camel case to underscore (in order to fit the search result to the local naming convention) when the clone was created, this adaption should be applied to any incoming change as well.

Search results need to be adapted in order to fit into the target code. Integrating code search results into a local code base is, by its nature, unanticipated code reuse and will thus naturally require adaptions to the external search results so that they fit into the local code base. Search-driven development refers to this as *suitability* of search results [2].

IV. Technical feasibility

In this section, we discuss how hot clones can be recorded upon creation and how they should be presented to the user. A key challenge is tracking code snippets while they evolve to become increasingly distinct. The versioning system can be informed of the original cloning and henceforth try to keep track of the clone automatically, even if it is not marked in the source code.

Figure 1 illustrates a sketch of the proposed prototype. We plan to extend MONTICELLO, a distributed version control system for Smalltalk code, with two components: a code search engine and a clone repository.

The *code search engine* provides access to the full content of the http://www.squeaksource.com installation of Monticello, some 7.5 GB of Smalltalk source code[1]. On the client side, the search engine allows the developers to create semantically transformed clones that suit the local code base. Each method that belongs to the clone is tagged with the unique identifier of the clone.

The *clone repository* adds a commit hook to the MONTICELLO version control system that is triggered whenever someone commits a new version of a method that has (or previously had) a clone id. The hook adds the client to the list of linked clones (if not already present) and informs all linked clones of the update through an RSS feed. Clients that are subscribed to the RSS feed can present their developers with an option to update their instances of the clone in question. Also, the Pharo IDE can be extended to show all clones of a method from the context menu of that method.

V. Obtained results

Since our use case suggests large amounts of source code to deal with, we proposed a set of lightweight techniques that may scale up to very large amounts of source code in the presence of multiple versions. The common idea behind these techniques is to use bad hashing to get a quick answer. By bad hashes we mean those that map different — but similar — code snippets to the same hash. Bad hashing can be used to detect cloning events simply by searching for identical bad hashes in any observed repository.

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.

Once similar snippets have been detected in two projects, the snippet needs to be tracked. Currently, we use a very simple metric: if the same snippet is discovered in two different repositories, then a hot clone is created. All methods that contain the snippet are added to the hot clone. All future versions of the method, whether or not they contain the original snippet, are now tracked in the hot clone. Since this is possibly too inclusive, it is vital to give the user the option to cut off the tracking of hot clones.

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 as shown in Table I [13].

A. Future work

We will evaluate our definition of bad hashing and type-1, type-2 and type-3 clones against Bellon's benchmark [14]. Furthermore, we plan to put our techniques to the test by applying them to other large ecosystems such as the Maven

[1]From a tweet by Lukas Renggli, project lead of Squeaksource: "http://www.squeaksource.com hosts 7.5 GB of Monticello versions," posted on Jan 7, 2010, http://twitter.com/renggli/status/7473119028

Table I – Percentage of cloned methods and classes out of 560,842 methods and 74,026 classes on SqueakSource.

	Type 1	type-2	Type 3
Percentage of cloned methods	14.55 %	16.33 %	17.85 %
Percentage of cloned classes	0.16 %	0.19 %	0.21 %

repository. We have already generalized our clone detector to the Java programming language.

Even though bad hashing is a good starting point to limit the amount of work necessary to detect and track hot clones, a better way to store our data will be necessary. Our analysis and work with large amounts of data requires queries to be able to run in parallel. Thus, we are evaluating alternatives to storing our data in Postgresql.

Acknowledgments: We gratefully acknowledge the financial support of the Swiss National Science Foundation for the project "Bringing Models Closer to Code" (SNF Project No. 200020-121594, Oct. 2008 - Sept. 2010). We thank Oscar Nierstrasz and Edouard Tavinor for their ideas and corrections.

References

[1] R. Hoffmann, J. Fogarty, and D. S. Weld, "Assieme: finding and leveraging implicit references in a web search interface for programmers," in *UIST '07*. New York, NY, USA: ACM, 2007, pp. 13–22. [Online]. Available: http://dx.doi.org/10.1145/1294211.1294216

[2] 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

[3] 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

[4] 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

[5] 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

[6] 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

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

[8] 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

[9] N. E. Schwarz, E. Wernli, and A. Kuhn, "Hot clones, maintaining a link between software clones across repositories," in *Proceedings of the 4th International Workshop on Software Clones*, ser. IWSC '10. New York, NY, USA: ACM, 2010, pp. 81–82. [Online]. Available: http://dx.doi.org/10.1145/1808901.1808915

[10] 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

[11] G. Muller, Y. Padioleau, J. L. Lawall, and R. R. Hansen, "Semantic patches considered helpful," *SIGOPS Oper. Syst. Rev.*, vol. 40, no. 3, pp. 90–92, 2006. [Online]. Available: http://dx.doi.org/10.1145/1151374.1151392

[12] M. Denker, T. Gîrba, A. Lienhard, O. Nierstrasz, L. Renggli, and P. Zumkehr, "Encapsulating and exploiting change with changeboxes," in *ICDL '07*. New York, NY, USA: ACM, 2007, pp. 25–49. [Online]. Available: http://dx.doi.org/10.1145/1352678.1352681

[13] N. Schwarz, M. Lungu, and R. Robbes, "On how often code is cloned across repositories," Manuscript submitted for publication, 2011.

[14] S. Bellon, R. Koschke, G. Antoniol, J. Krinke, and E. Merlo, "Comparison and evaluation of clone detection tools," *IEEE Transactions on Software Engineering*, vol. 33, no. 9, pp. 577–591, Sep. 2007.