The Universal Repository of Everything

Niko Schwarz, University of Bern
Daniel M. German, University of Victoria
Serge Demeyer, University of Antwerp
Michael W. Godfrey, University of Waterloo
Douglas H. Martin, Queen's University

***Abstract*—Clone detection might strike the casual observer as a proverbial research solution looking for a problem. However, when carefully considered, it can be seen as comprising three closely related problems: clone detection, code diffing, and code search. Techniques proposed for any one of them can typically be used for the others as well. Together, these techniques are hugely successful both in research and practice. This article recollects success stories of the techniques, points out open problems, and suggests a vision of the whole that the techniques may be forming: the universal repository of everything.**

***Index Terms*—clone detection; software provenance;**

I. Introduction

Clone detection seeks duplication in programs, either exact or near-miss, [1] and has been studied heavily from a somewhat algorithmic point of view. There is a plethora of clone detection techniques with estimates on their precision and recall, as well as their performance figures and scalability.

The dual of clone detection is code *diffing*: the search for dissimilarities between programs. Since two snippets of a program are either similar or dissimilar, diffing and clone detection are two sides of the same coin.

Code search is somewhat broader than the previous two techniques in that the search term need not resemble the representation of the best find, it need only express the same concept. Since literal resemblance often implies conceptual resemblance, clone detection is one ingredient among many to good searching.

In this paper, we recollect success stories of the above three problems and the techniques used to solve them, and we suggest a vision of the whole that the techniques may be forming: the universal repository of everything. By this we mean a dictionary that maps the signatures of all artifacts ever encountered to the place they were encountered in.

A. The Universal Repository of Everything

Many of the techniques that we discuss in this article follow the same underlying principle: an artifact is reduced to a signature, which is then compared to other signatures in order to find similar artifacts from elsewhere. Since every artifact has at least one binary representation, the simplest signature is the 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 that aren't important for overall similarity.

This leads us to the vision of the universal repository of everything. It stores the signatures of all digital artifacts ever produced, in all of their versions. It maps from the signature to a descriptor of the origin of the artifact. This allows to track the divergent evolution of software artifacts [10], as well as establish the provenance of every artifact encountered. We believe there are countless possible applications. As a rule of thumb, with massive amounts of data to help, difficult problems can suddenly become a lot easier [6].

While our repository might never be realized concretely, we note that since signatures are fixed in size, the size of our repository is linear in the number of all artifacts ever produced, which may already be feasible from a memory and performance point of view (for workable definitions of "artifact").

The open problem to produce the universal repository of everything is a working catalog of techniques that produce signatures that reflect just the amount of similarity that is needed. In the rest of this paper we will look at current techniques that abstract artifacts, and we will see applications of these abstractions.

II. Clone Detection in Binaries

Clone detection is ordinarily performed on program sources, but can also be run on binaries. Depending on whether the binary analysis is assumed to be adversarial (*i.e.,*, with the intent to hide the underlying implementation), different abstractions may be employed. For non-adversarial clone detection, a simple technique has been shown to be surprisingly successful: programs are scanned for the frequency of certain markers. In the detection of violations of the GPL license in firmware binaries, the search for strings in binaries has been shown to be highly effective [7].

Beyond license violations, looking up a binary in a global repository of binaries could reveal which version of the binary it is—a question Java developers seem to be asking about their binaries with astonishing frequency. In some cases, a global repository of hashes of binaries can identify the correct version. However, in practice, there is often more than one binary in circulation for any given version, due to the choice of compiler or compile-time options selected. A number of markers have been proposed to allow a fuzzy search for Java binaries [3], [4], which focus on the underlying version of the jar, but ignore the compiler that was used to produce it.

Computing the differences between binaries is important to the security of software systems. When a system gets patched, the patch necessarily contains information on the vulnerability that was fixed. Since not all users of software will immediately perform an update, it is important to software providers to make it difficult for their binaries to be diffed, in order to protect the yet unpatched installations. Conversely, developers want to know where their systems are vulnerable.

Open Questions: For two binaries, can we answer whether one of them is a descendant of the other? How can code duplication tools on binaries be benchmarked?

III. Provenance

Provenance, in general, refers to the record of origin and ownership of an artifact. In the software engineering world, the problem of provenance of a software artifact refers to the recovery of its origin or source. The Internet has made it easy to copy source code (particularly open source). Unfortunately, version control systems only start recording the history of an artifact the moment that it is a added, and does not link it to its original source. The question of provenance arises naturally in the following domains.

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: When supplied with both source code and binaries, one 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.

The wish for descriptive meta information on software has led to the creation of the Software Package Data Exchange format [9], which marks every software module with where it came from, what the license is, and who developed it, much like the sticker on a ship container, or a hardware bill of materials listing every screw. Currently, ad hoc methods prevail in the search for provenance of source code.

There are several techniques that try to solve this problem on the level of the build system or package manager [5]. These methods record and track identifiers in the software artifact, albeit with limited success.

Open Questions: What type of information is present in the source that is preserved in the binary?

IV. Non-code clones

The artifacts in modern software projects include more than source code. This leads to a problem: while source code is nearly universally stored in plain text, other artifacts like UML diagrams can be stored in a variety of formats. We may still be able to use existing techniques on these other formats if we manipulate them to find the right granules [8].

To prevent and punish copyright violations, copyright holders are scanning the Internet for similar copies of their pictures. Websites that host copyrighted material, to be protected by "safe harbor" laws, are now offering the search for "clones" in images and videos.

Open questions: How can images be reduced to signatures, so that similar pictures still produce the same signature?

A. Clones in Bug Reports

A software systems with a large user base receives a huge amount of bug reports. For instance, projects like eclipse, GNOME and Mozilla received between 2500 and 6900 bug reports over the course of three monthsMany of these bugs are duplicates; Eclipse for instance identified 20 % duplicates or on average 371 duplicates per month [2]. As such, finding duplicate bugs has received considerable attention in recent years and several techniques such as text mining, or matching stack traces, have been explored with reasonable success.

Open Questions: Can we see if the bug was already reported to a subcontractor? Can we identify which component releases are more trustworthy than others?

V. Summary

We have proposed the universal repository of all artifacts of software engineering ever produced. We identified open question on our quest towards a catalog of techniques that allow us to store signatures of all kinds of artifacts in one repository.

Acknowledgment

We gratefully acknowledge the financial support of the Swiss National Science Foundation for the project "Synchronizing Models and Code" (SNF Project No. 200020-131827, Oct. 2010 - Sept. 2012). This paper comes from Schloss Dagstuhl Seminar 12071 - Software Clone Management Towards Industrial Application. We thank Armijn Hemel and Jens Krinke for their contributions.

References

[1] Ira D. Baxter, Andrew Yahin, Leonardo Moura, Marcelo Sant' Anna, and Lorraine Bier. Clone detection using abstract syntax trees, 1998.

[2] N. Bettenburg, R. Premraj, T. Zimmermann, and Sunghun Kim. Duplicate bug reports considered harmful... really? In *Software Maintenance, 2008. ICSM 2008. IEEE International Conference on*, pages 337 –345, 28 2008-oct. 4 2008.

[3] Julius Davies, Daniel M. German, Michael W. Godfrey, and Abram Hindle. Software bertillonage: finding the provenance of an entity. In *Proceedings of the 8th Working Conference on Mining Software Repositories*, MSR '11, pages 183–192, New York, NY, USA, 2011. ACM.

[4] M. Di Penta, D.M. German, and G. Antoniol. Identifying licensing of jar archives using a code-search approach. In *Mining Software Repositories (MSR), 2010 7th IEEE Working Conference on*, pages 151 –160, may 2010.

[5] Eelco Dolstra. *The Purely Functional Software Deployment Model*. PhD thesis, Utrecht University, January 2006.

[6] Alon Halevy, Peter Norvig, and Fernando Pereira. The unreasonable effectiveness of data. *IEEE Intelligent Systems*, 24:8–12, 2009.

[7] Armijn Hemel, Karl Trygve Kalleberg, Rob Vermaas, and Eelco Dolstra. Finding software license violations through binary code clone detection. In *Proceedings of the 8th Working Conference on Mining Software Repositories*, MSR '11, pages 63–72, New York, NY, USA, 2011. ACM.

[8] Douglas Martin and James R. Cordy. Analyzing web service similarity using contextual clones. In *Proceedings of the 5th International Workshop on Software Clones*, IWSC '11, pages 41–46, New York, NY, USA, 2011. ACM.

[9] Phil Odence and Kate Stewart. A common software package data exchange format:1.0 release update and discussion, August 2011.

[10] Nikolaus E. Schwarz, Erwann Wernli, and Adrian Kuhn. Hot clones, maintaining a link between software clones across repositories. In *Proceedings of the 4th International Workshop on Software Clones*, IWSC '10, pages 81–82, New York, NY, USA, 2010. ACM.