Similarity management of 'cloned and owned' variants*
(Extended Abstract)

Thomas Schmorleiz Ralf Lämmel
Software Languages Team, University of Koblenz-Landau

Abstract

The 'clone and own' approach to software product lines assumes that variants are created by cloning and evolve more or less independently afterwards. We research the process to manage similarity of such 'cloned and owned' variants along the timeline. The process uses annotations for recording developer intentions and it leverages automatic change propagation. We have applied the process and corresponding tool support in a case study where we manage similarity for clowned-and-owned Haskell-based variants of a simple human-resources management system.

1 Introduction

The discipline of *clone management* studies detection, avoidance, and removal of clones as well as compensating activities. For instance, such compensation helps in the context of the 'clone-and-own' approach to managing *software variants* [7, 8, 4, 1]; the approach is widely established in industry for reasons such as developer independence [8, 2]. In our approach, we address the clone-and-own approach by a combination of variability and clone management which we refer to as *similarity management*. We treat the variants as a cloning genealogy [5, 9]. Clone detection is performed at a granularity level of 'fragments' by which we mean named abstractions such as methods or functions. Similarity measures for source-code files, folders, and variants enable exploration of similarity at different levels and, ultimately, similarity improvement. The developer must decide whether a found similarity is to be maintained or improved. In the first case, we speak of a *maintenance invariant*; in the second case of a *maintenance task*.

Summary of the research a) We introduce similarity measures and classifiers to capture similarity and evolution thereof across clowned-and-owned variants. b) We present a process for similarity management with activities for analyzing, annotating, maintaining, and improving similarities. c) We devise designated tool support. d) We present a first case study on similarity management.

All tools and artifacts underlying the case study as well as the full paper are available online[1].

Research question How to measure similarity of clowned-and-owned variants and effects of similarity improvement in a useful manner?

One can define similarity at the fragment-level in some standard way using algorithms such as longest common subsequence or Levenshtein distance. However, we seek a deeper understanding of similarity including the development of fragment sharing on the time line. We answer the question by researching metrics such as the number of distinct fragments across all variants and the number of variants a fragment is shared in.

2 A process for similarity management

Analysis An automated analysis is applied to the commit history of given variants, thereby determining how fragment-level similarities evolved over time. For instance, a similarity may be identified as having 'diverged from equality'. The same analysis is applied incrementally whenever the developer makes local changes or pulls new versions.

Review Subject to interactive tool support, the developer reviews similarity evolutions sorted by decreasing similarity measures, i.e., equalities show up at the top. Similarities that are seen as adequate may be tagged to be taken as invariants such as 'Maintain Equality'. Similarities that have evolved unreasonably may be acted upon as follows.

Improvement Option 1: The developer suggests *automated change propagation* to make two fragments the same again. As a side effect, the resulting fragment equality is tagged as an invariant that is to be maintained. Option 2: The developer *manually changes* the involved fragments to increase similarity, thereby, again, implicitly tagging the result as an invariant. Option 3: The developer *defers the change, but tags* the similarity evolution at hand with a maintenance task for a due improvement such as 'Increase Similarity'.

Reestablishment Subject to interactive tool support, the developer is notified of invariants that have

*The full paper appeared in the proceedings of SAC 2016.

[1] `http://softlang.uni-koblenz.de/simman`

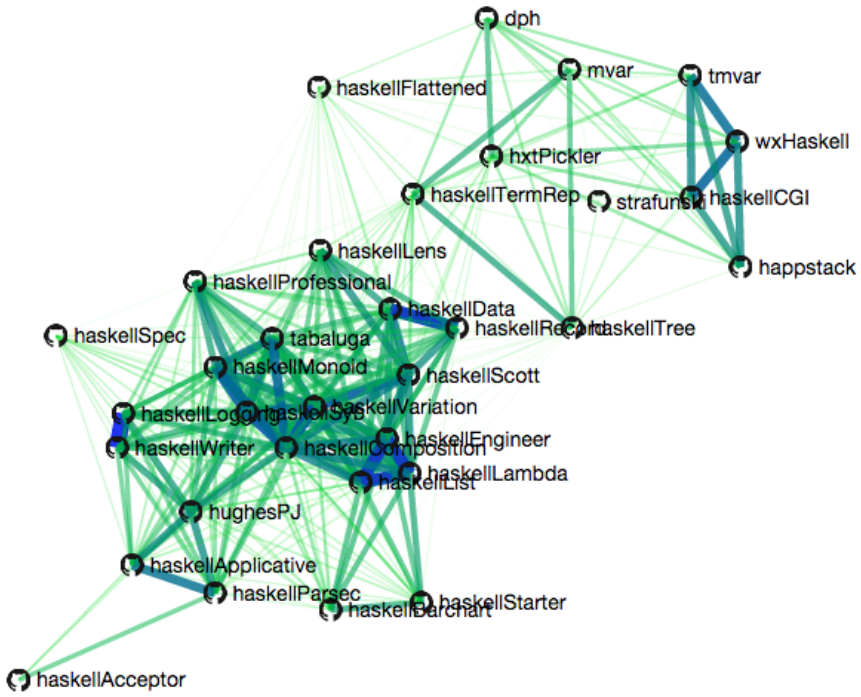

Figure 1: Force-driven layout of the variants in the case study: proximity correlates with degree of similarity.

been broken by local or pulled changes, thereby encouraging the developer to reestablish these invariants by means of improvement, as discussed before, or to possibly abandon them.

Committal The developer commits the local version also including the annotations for invariants or pending maintenance tasks. Thereby, developers can manage similarity collaboratively.

3 Case study

We manage similarity for *101haskell* [6] which is a set of 36 Haskell-based variants of a small human-resources management system; *101haskell* is part of the *101companies* chrestomathy [3]. Fig. 1 visualizes the similarity of the variants in the case study based on the aggregation of fragment-level similarities to the variant level. We dealt with a commit history with 961 versions, developed over 42 months. We made 345 annotations triggering 448 maintenance tasks: 324 automated and 124 manual changes. Process execution required only a few hours of systematic work. Similarity improvement would have been very tedious and time-consuming without the process and designated tool support. Also, the established maintenance invariants help maintaining similarity in the future while also relying on change propagation.

References

[1] Michal Antkiewicz, Wenbin Ji, Thorsten Berger, Krzysztof Czarnecki, Thomas Schmorleiz, Ralf Lämmel, Stefan Stanciulescu, Andrzej Wasowski, and Ina Schaefer. Flexible product line engineering with a virtual platform. In *Proc. of ICSE 2014*, pages 532–535. ACM, 2014.

[2] Yael Dubinsky, Julia Rubin, Thorsten Berger, Slawomir Duszynski, Martin Becker, and Krzysztof Czarnecki. An exploratory study of cloning in industrial software product lines. In *Proc. of CSMR 2013*, pages 25–34. IEEE, 2013.

[3] Jean-Marie Favre, Ralf Lämmel, Thomas Schmorleiz, and Andrei Varanovich. 101companies: A Community Project on Software Technologies and Software Languages. In *Proc. of TOOLS 2012*, pages 58–74. Springer, 2012.

[4] Stefan Fischer, Lukas Linsbauer, Roberto Erick Lopez-Herrejon, and Alexander Egyed. Enhancing clone-and-own with systematic reuse for developing software variants. In *Proc. of ICSME 2014*, pages 391–400. IEEE, 2014.

[5] Miryung Kim and David Notkin. Using a clone genealogy extractor for understanding and supporting evolution of code clones. *ACM SIGSOFT Software Engineering Notes*, 30(4):1–5, 2005.

[6] Ralf Lämmel, Thomas Schmorleiz, and Andrei Varanovich. The 101haskell chrestomathy: A whole bunch of learnable lambdas. In *Proc. of IFL 2013*, pages 25:25–25:36. ACM, 2014.

[7] Julia Rubin and Marsha Chechik. A framework for managing cloned product variants. In *Proc. of ICSE 2013*, pages 1233–1236. IEEE, 2013.

[8] Julia Rubin, Krzysztof Czarnecki, and Marsha Chechik. Managing cloned variants: A framework and experience. In *Proc. of SPLC 2013*, pages 101–110. ACM, 2013.

[9] Shuai Xie, Foutse Khomh, and Ying Zou. An empirical study of the fault-proneness of clone mutation and clone migration. In *Proc. of MSR 2013*, pages 149–158. IEEE Press, 2013.