Assessing Cross-Project Clones for Reuse Optimization

Veronika Bauer, Benedikt Hauptmann
Technische Universität München, Germany
{bauerv, hauptmab}@in.tum.de

***Abstract*—Organizational structures (*e. g.*, separate accounting, heterogeneous infrastructure, or different development processes) can restrict systematic reuse among projects within companies. As a consequence, code is often copied between projects which increases maintenance costs and can cause failures due to inconsistent bug fixing. Assessing cross-project clones helps to uncover organizational obstacles for code reuse and to leverage other ways of systematic reuse. Furthermore, knowing how strongly clones are entangled with the surrounding code helps to decide if and how to extract them to commonly used libraries. We propose to combine cross-project clone detection and dependency analyses to detect (1) what is cloned between projects, (2) how far the cloned code is entangled with the surrounding system and (3) what are candidates for extraction into common libraries.**

***Index Terms*—Clone detection, cross-project, code reuse**

I. Introduction

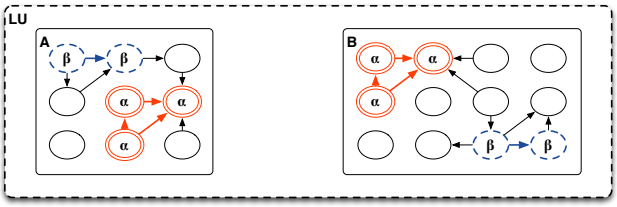
Companies developing software systems for a specific application domain are often confronted with recurring implementation tasks. Since a lot of domain knowledge is manifested in the code, it suggests itself to reuse successful solutions accross project boundaries. However, organizational factors can restrict systematic reuse among projects. For example, separate accounting of charges, different development models or heterogeneous infrastructure can prevent teams to create commonly accessible, reusable, software modules.

Consequently, cloning is a popular way to reuse sucessful implementations across project boundaries [1]. Especially if a developer implemented a solution in a previous project, (s)he is likely to copy and adapt it. According to [2] as well as to own experience, these cross-project clones (CPCs) tend to be larger portions of code, ranging from several methods to files or entire subsystems. This "simple" way of overcoming reuse difficulties is known to have negative consequences on maintenance costs and software quality in the long term [3].

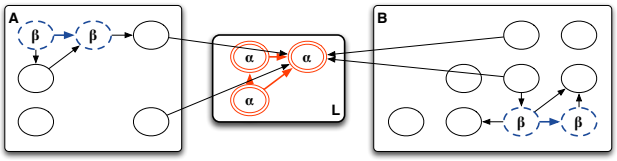
We propose to combine cross-project clone detection (CPCD) with static dependency analyses to assess and manage this aspect of code reuse with the goal to provide input for organizational and maintenance decisions concerning reuse.

II. Idea

We would like to answer the following questions: To which amount does cross-project cloning (CPC) exist among a specific set of projects (within a company, a division, of a certain technology etc.)? Do cross-project clones (CPCs) implement specific functionality? Are there CPCs that would qualify as candidates for library creation? If yes, which of the CPCs could be extracted with reasonable effort? What are the organizational reasons that give rise to CPC?



(a) Cross-project clone detection and dependency analysis



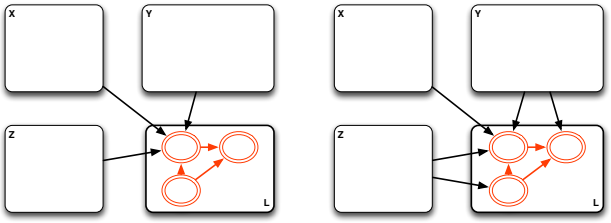
(b) Goal: Extraction of candiates in library

Fig. 1. The figure illustrates the three steps of our idea. A and B denote two software systems. LU denotes the logical union of the systems during clone detection. Circles represent entities of the systems, e.g. functions, classes, or files. The systems contain two clone classes: the double-lined (α) and the dashed (β) group. Arrows denote the dependencies between entities belonging or connected to clones. L denotes an extracted library, containing (α).

First, we run a clone detection on all systems of interest, treated as one logical unit (*LU* in Figure 1(a)). We filter the findings to CPCs. The result already provides relevant information: the extent of cloning within systems compared to the extent of cloning accross systems and the total extent of CPC. Therefore, we can quantify CPC at this point.

Next, we aim to detect clone classes potentially implementing coherent functionality. We attempt this by identifying clones that form weakly connected components in the dependency graph. We argue that this is reasonable as own experience as well as existing work (c.f. [2]) have indicated CPCs to be rather large, potentially comprising several functions, classes, or even entire system components. We expect most of these large chunks of code to be reused in a library-style manner, *i. e.*, being called from the surrounding system parts, but not calling back into the system. Based on these assumptions, we propose to run a static dependency analysis to identify clones that are candidates for extraction.

Accepted for publication by IEEE. © 2013 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/ republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.



(a) Dedicated point of entry (b) Scattered access

Fig. 2. Different usage patterns of cloned regions.

To achieve a sufficient precision, the analysis works on a method granularity. For each clone class, we assess all instances, as usage patterns might differ. For each clone, we start from its location in the code. On the dependency graph, we trace all outgoing calls until we reach the border of the cloned region. We then assess the number of calls from the cloned region into uncloned system code. Ideal extraction candidates do not call into the system (see clone α in Figure 1). Clones with a large number of calls into system code are less desirable for extraction (see clone β in Figure 1). In practice, we expect calls into the system to happen and propose to use a threshold value for the number considered acceptable[1].

From the clone-system border in the dependency graph, we trace the calls from the system into the cloned region. In this way, we can collect usage profiles of the clones. Figure 2 shows two examples of potential usage patterns. Comparing the calling patterns within the same clone class allows to deduce requirements for an extracted library.

The last step is to assess the functionality implemented by the extraction candidates. The process of functionality assessment serves as input for library creation. Figure 1(b) illustrates the goal of our idea: the candidate has been extracted into a new library, which both of the systems call.

III. Usage Scenarios

Reuse assessment: Being able to quantify the extent of cloning between projects enables to understand dimension and causes of reuse by cloning. For example, the existing technical infrastructure might hamper sharing code between teams. Improving the development infrastructure would enable developers to use structural reuse methods more often.

Identify demand for certain libraries: To make a library appealing for developers, it should provide a common set of helpful solutions for a certain class of problems (such as I/O, printing or DB access). Being able to identify cloned code helps to characterize the reused functionality. Thereby, the demand for solutions for a class of problems can be determined. For example, if I/O handling code is among the most frequently cloned functionality, a commonly accessible library providing I/O functionality would offer developers an alternative to cloning code between projects.

[1]Arguably, very prominent but strongly connected clone groups should also be assessed manually.

Assess effort for code extraction: Knowing how far cloned code is entangled with the surrounding system helps to estimate the complexity to separate it by creating a library. If, *e. g.*, cloned code is strongly interwoven in several projects, extracting the code might require exorbitant ressources compared to the resulting savings in maintenance effort.

IV. Related Work

Cross-project clone detection Mende et al. [4] propose CPCD to support the grow-and-prune model [5] for Software Product Lines management. Schwarz et al. [6] provide evidence of a significant amount of CPC for the Smalltalk ecosystem. Furthermore, they propose scalable clone-detection techniques for CPCD. Krinke et al. [2] assessed the copying and cloning between projects of the GNOME Desktop Suite, studying the flow of code between the different projects. Al-Ekram et al. [7] investigate cloning "by accident", a consequence of projects implementing identical code portions. Instances could be candidates for inclusion into libraries.

Higher-level clones Basit and Jarzabek introduce structural clones [8], logical groups of simple clones. We follow this idea to better identify code portions with coherent functionality that are candidates for extraction into libraries.

Assessment of reuse In earlier work [9], [10] we proposed an assessment model for usage of third-party libraries as well as code reuse within organizations. CPCD will be integrated as one measure for the maturity of code reuse.

V. Next Steps

The next step is to implement the proposed idea, building on the clone detection and dependency analyses provided by the ConQAT[2] toolchain. To assess the benefit for development and maintenance in practice, we will apply our idea on a set of projects of an industrial partner. Furthermore, we will integrate the results in the assessment framework for code reuse, which we are currently developing.

References

[1] Y. Dubinsky, J. Rubin, T. Berger, S. Duszynski, M. Becker, and K. Czarnecki, "An exploratory study of cloning in industrial software product lines," in *CSMR 2013*, 2013.
[2] J. Krinke, N. Gold, Y. Jia, and D. Binkley, "Cloning and copying between gnome projects," in *MSR'10*, 2010.
[3] E. Juergens, F. Deissenboeck, B. Hummel, and S. Wagner, "Do code clones matter?" ser. ICSE '09, 2009.
[4] T. Mende, F. Beckwermert, R. Koschke, and G. Meier, "Supporting the grow-and-prune model in software product lines evolution using clone detection," in *CSMR 2008*, 2008.
[5] D. Faust and C. Verhoef, "Software product line migration and deployment," *Software Practice and Experience*, 2003.
[6] N. Schwarz, M. Lungu, and R. Robbes, "On how often code is cloned across repositories," in *ICSE'12*, 2012.
[7] R. Al-Ekram, C. Kapser, R. Holt, and M. Godfrey, "Cloning by accident: An empirical study of source code cloning across software systems," in *ISESE'05*, 2005.
[8] H. A. Basit and S. Jarzabek, "A data mining approach for detecting higher-level clones in software," *IEEE Trans. on Software Engineering*, 2007.
[9] V. Bauer, L. Heinemann, and F. Deissenboeck, "A Structured Approach to Assess Third-Party Library Usage," in *ICSM'12*, 2012.
[10] V. Bauer, "Facts and fallacies of reuse in practice," in *CSMR 2013*, 2013.

[2]www.conqat.org