
Abstract
ET: Enhanced tool
The effectiveness of <clone detection tools> in supporting the design of <maintainable code bases> has [perhaps] been demonstrated. An enhanced tool is described for the design of <code bases> based on <depicting the genealogical relationships among code clones>. Examples are provided confirming the effectiveness of its support for <analyzing whether code clones really do harm the maintainability of code bases> in design [and we think that they don’t!]

Why ET?
This one was harder to categorize than usual. It seems that their main deliverable is to extend existing clone detection tools to make a new tool that depicts clone genealogy. However, the main implications that they learned from using the tool seem to be that we shouldn’t automatically assume that cloning code is bad, which has implications for anybody who wants to build models of code quality. I’ll assume that the tool was the contribution here, since that was a deliverable, even though model-building researchers may want to take note of this paper.

Question - [Method/means of development]
How can we automate the analysis of how code clones are created and evolved?

Results - [Tool / Notation]
The authors extended existing clone detection software to create a tool that depicts the common parentage of clones, their evolution, and their death. They noted that in the examples discussed below that many clones appeared be unfactorable (meaning that the language strongly inhibited creation of abstractions to prevent the cloning) and that many clones died out (suggesting that programmers were “trying out” a strategy before setting on an abstraction).

Validation – [Examples]
The authors test drove their new tool on two open source projects, dnsjava and carol.

In terms of internal validity, the authors didn’t provide enough information to convince me that their tool really did represent genealogies. Part of that was because the underlying clone detector tools have a very fuzzy edge as to what is a clone and what isn’t. I was also put off by the large number of clones that the authors manually removed from the data set. Finally, even if one code snippet was created immediately after another, and the snippets are substantively similar, that doesn’t necessarily mean that they have the same semantic purpose, and CVS data cannot address this question. For example, if I make a copy of a co-worker’s code, I might just be using it as an example, with no intention whatsoever of keeping it consistent with his—and the authors data suggest that this is not an uncommon situation.

In terms of external validity, I doubt that their findings could automatically be generalized to commercial software development. Many companies have a code base and mandate that employees reuse code from the base rather than making clones. At the companies where I’ve worked, we would actually get in trouble if our bosses found out that we were reinventing the wheel! This could have significant sociological implications that might generate different results than those found in this paper. For example, clones might turn out to be much less common, or perhaps the relationship between clones is much more subtly achieved by programmers, making it harder for the tool to detect genealogical relationships.

Finally, do they know that code clones really don’t hurt maintainability? That was the question in the first place. But the authors really never come back to this external validity question in a compelling way. Effectively, they just say, “It looks like some code clones are unavoidable, and many code clones die out anyway,” which from an external validity standpoint is unsubstantiated.