Review 12

3/1/12

CS6V81.502

Seungtack Baek

**An Empirical Study of Code Clone Genealogies**

In this paper, Kim et al. used code clone genealogies to find that

1. Difficulty code clones impose on programmers.
2. The lifespan of code clone (as in number of revisions it survived) and
3. How does clone change throughout its genealogy.

Code genealogy is composed of one or more code lineage. Code lineage is a model where it tracks groups of code clones whenever it undergoes certain changes, such as addition, subtraction, shift and consistent/inconsistent change. These changes can overlap.

The researchers applied this code genealogy to two java projects to answer above questions. As an answer to the first question, they found 36%-38% of the code clones to be constantly changing which means programmers had to be constantly bugged by code clones. Then to find out the average lifespan of code clones, they classified code groups into two categories: dead or alive (since you cannot tell the longevity of both 3 year old boy of 80 year old grandpa). They found that about 31%-37% of genealogies have died within 5 revisions. This result is more than I expected it to be, since my personal experience was that I would make code clone, unless it is unavoidable and most of the time, it is unavoidable until the production phase. Yet, this study’s claim seems viable for the fact that programmers might find some ways to “abstractize” some of the intersections of code clone (which doesn’t really make sense to have “intersection” for the clones… Aren’t they supposed to be the intersection themselves?).

Also they came up with new clone group, “locally unfactorable,” where refactoring needs to be applied not only to the module itself, but also to the related modules, such as its implementors or its childrens. The other type of clone group is “Long-lived” group. The advantage of this group underlies in that it actually worth for programmers to maintain such group, without wasting their time trying to fix goner clones.

The author frequently relates code clone with “refactoring.” Yet, while I was reading this, I found out just how the word “refactoring” is as vague as “change.” I wish they have been more specific, such as “variable name change,” etc. Also, just as much as “refactoring,” “clone” itself is also really hard to define. How can you tell if two pieces of code are similar? They came up with equation for TextSimilarity(t1; t2) and LocationOverlapping(Li; Lj ), assuming that it works. I would like to see how good each measurement is.

Kim et al. proposed that code genealogy is beneficial in clone management. I would also like to see some more feasibility evaluation of code genealogy, such as performance of the algorithm, etc.

**Question:**

1. This goes back to “how to write good research paper.” But, would you consider the first sentence of second paragraph in introduction as good example of citation? (it lists out all relevant sources, yet I think it is suitable in this case, if all 9 works focused on automatized code clone detection.)
2. What is “syntactic sugar?” (p.3 third sentence from the last in the last whole paragraph in the first column)