Estimating Coder Productivity

The idea in this algorithm is to estimate programmer productivity by considering two loosely coupled variables — (i) number of commits over time and (ii) number of line-changes over time. See the API endpoint from GitHub below, which supplies these stats:

<https://developer.github.com/v3/repos/statistics/#get-the-number-of-additions-and-deletions-per-week>

Let *C* be a function of variable (i), the # commits over one week..

Let *L* be a function of variable (ii), the # line-changes over one week.

Then for each week, our estimate of programmer productivity λ is the weighted sum:

*λ = 0.5\*C + 0.5\*L*

λ should fall in [0,1]. When it is 0, then a programmer has done nothing. When it is 1, the programmer is at peak productivity. We may adjust scaling factors here later.

Now, the goal is to define functions *C* and *L*.

*C* should factor in both long-term and short-term behavior. Short-term behavior is simply how a programmer (in terms of weekly # commits) compares to “everything else” that happened; for example, if for some week there are a total of 57 commits, and programmer P has 15 commits, then his short-term behavior is captured by the fraction 15 / 57. Long-term behavior is more heavily computational — here, we compute a histogram that says what frequency counts are for # commits per week. The histogram is calculated over duration of project. Long-term behavior is simply an integral or sum — add up all frequency counts less than the current # commits for P this week, and divide by total frequency counts. If we cannot get a histogram for duration of project, the last 30-45 days is ok.

Call short-term behavior *CS*. Call long-term behavior *CL*. Then *C* will be a weighted sum like this:

*C = 0.5\*CS + 0.5\*CL*

So, *C* should fall in [0,1].

I think we can similarly define *L*. That is, *L* should comprise both long- and short- behaviors:

*L = 0.5\*LS + 0.5\*LL*

where *LS*says how performant a programmer is versus “everything else” that happened in some given week, while *LL*is a statement about a programmer’s behavior versus the line-change history of the project’s entire (or abbreviated) lifecycle.

*Charles Read; March 15, 2017.*