CS 6704: CTTC 5

Sourabh Shetty

SOTorrent: reconstructing and analyzing the evolution of stack overflow posts

I agree with the authors' decision to exclude URLs within code blocks from their list of external links. Code, especially when copied and pasted or autogenerated, frequently has links in comments, or contains links that the code is meant to connect to or refer from, which don't actually have a relation or give further context to the code itself. So it makes sense to exclude them.

I however disagree with their decision to not include inline code while counting code blocks for changed code. Inline code can be just as important as continuous code snippets and are just as likely to be edited and updated. If a piece of code is small enough, creating an entire new code block can seem like overkill and so inline code can be a good way to include that code, even in the middle of a sentence. But even that inline code can be updated. We see this frequently in cases where Python 2 code examples from years ago is updated with Python 3 code now.

I also like the idea of analyzing if the code had been edited after a comment had been made. While they did do some research on it, I think this could lead to very interesting results. Especially if they can take into account how many points a comment has, since often high voted comments on answers prove to be very important.

A Study on the Use of IDE Features for Debugging

I very strongly disagree with how the authors here have calculated the debugging period. They appear to be using test failures as their primary metric for detecting a debugging session. While ideally there should be test cases for everything and that's ideally how things should always run, that isn't the case most of the time. There aren't tests in place for every possibility, and when the code is buggy, it's often identified at runtime rather than by failing test cases.

The authors dismiss the notion that users may first try to debug their code without additional tools at first and then if switch to debuggers if that doesn't yield promising results. They dismiss this on the basis of the fact that users switched to debuggers within 5-13 minutes in most cases. I don't believe that changes anything here. 5-13 minutes is plenty of time for a user to realize that his chosen method of manual debugging might not be comprehensive enough to actually fix the issue and that it might be time to switch to a proper debugger. If anything, it actually supports the original assertion itself.

How Does Contributors' Involvement Influence the Build Status of an Open-Source Software Project?

It was surprising how surprised the authors were that casual contributors weren't causing as many failing builds. In fact they even claimed it as a discovery that went against what they assumed would be the prevailing thought. But the fact that the commit went through and a code integrator likely reviewed or tested the code before merging the pull request should have been a good enough indicator that the casual contributors code wouldn't fail as much. In fact, a casual contributor would likely be more careful while committing code, and the review process would be much more heavily scrutinized.

The authors also say that the size of the casual contributors' code being similar to the size of regular contributors code could be attributed to the casual users possibly simply adding to documentation or assets rather than actual code. To that, I say, doing some additional research and verifying that would not have been too hard. Checking whether code contribution was done in a documentation file or simply added asset files is not a particularly hard task and if it was a concern, it could have been researched.

Judging a commit by its cover: Correlating commit message entropy with build status on Travis-CI

I found this to be a very interesting topic to read about, but on the whole I did not expect any different from the results. While there is an expectation of clear, well written commit messages in big, respectable companies, the open source community has a will and mind of its own. This leads to a certain degree of informality even in large and popular repositories. But this informality doesn't mean the code quality will be bad or that the build will fail because of it. That being said, I don't disagree with any aspect of the authors research, because doubting such commits is very natural and so I find that having this research is actually quite valuable.