Affect Analysis

Jonathan Dreyer, Horia Mut, Valentin Py

***Abstract*—Modern application development requires the developers to often release commits in project in which many people collaborate. It might be difficult for the project leader to determine which pull request should be accepted or rejected in the project, regarding the quality of the modification of the program.**

In this paper we present the state of the art of emotion analysis of pull requests, then we describe how we implemented an automatic tool which parses pull request's associated emotions in order to help to quickly determine the quality of a pull request, based on other developers reviews. We then explain how we have implemented continuous integration and continuous delivery for this project.

Keywords*—*Pull requests analysis, Developer emotions, Project review

I. Introduction

In 2017, many open source projects are hosted on GitHub. Many known organizations share their projects under the open-source clause and are open to contributions. There are also self-hosted platforms that give developers the same functionality as GitHub except that this would be for in-house private use.

When a developer wants to contribute a certain feature or fix to a project, he will *push* his modifications into a *feature branch* and then generate what we call a *Pull Request*. This *Pull Request* should be reviewed by the maintainer of the repository, and then merged into the *master* or *release branch*. This is the normal workflow for software development when using a version control system (*VCS*). The difficulty lies in the review process of the *Pull Request*, especially when there are a lot of *commits* and the code-base is large within that *Pull Request*. A *commit* is a certain stage of the development process and it may or may not be functional.

In the following sections, we will describe our approach into easing the review process of a *Pull Request* by doing a quick guess of the *Pull Request*'s author emotional state.

II. Goal and Issue

Reviewing a *Pull Request* is hard work that needs to be done by the *repository*'s maintainer. This may be due to the fact that the code base is very large, the number of changes is also significant, and there are a lot of *commits* in the *Pull Request*. It can also be due to other external factors that affect the way the author of the request writes code. He/she may be in an emotional state that changes the quality of the code written.

Luckily, we have some features that are implemented on GitHub that can help us. On GitHub, each *commit* and *Pull Request* can be commented by any collaborator of the project. The comments can contain text, images, graphs, formulae, links and even ***emoticons***.

It is guessed that developers (and any person actually) that find themselves in a bad emotional state have a higher chance of writing bad code. Of course, they are not doing this on purpose but the emotions they feel subconsciously have a direct impact on the quality of the code they write. Since they are able to write comments and each commit has a mandatory description, we are assuming that the emoticons they use within these comments are written subconsciously as well and therefore may reflect the current emotional state.

Despite many researches, we have not found any solution that mixes code quality with the emotions of its authors.

III. Proposed Solution

We use these ***emoticons*** to classify each *Pull Request*. Of course, each emoticon must itself be classified beforehand. Without taking the context of use, a smiley would be a positive emoticon and a sad smiley would be a negative one. Of course, context may change the meaning. Currently, without applying machine-learning algorithms to sense the emotions within the text of the comments, it is impossible to detect context-sensitive emoticons. For example, sarcasm could see the laughing smiley used in a negative way. We have not taken context into account however this is a perspective open for future iterations.

Our solution therefore parses each of the author's *commits* of a *Pull Request* and wages the different *emoticons* present. A result is shown indicating the number of positive emotions and negative ones.

The reviewer can easily know if the Pull Request has a chance of containing bad-quality code or not at a quick glace of these results.

IV. Drawbacks of the proposed solution

While the guess that emoticons are mostly used to express what we feel subconsciously seems accurate, they may also be used with different purpose. The number of these emoticons should also be taken into account and for the moment it is not. Again, context should actually determine the correct classification of an emoticon.

The most problematic issue we face is that emotions may be suppressed by the developer when writing. While it can still affect his/her judgment, the comments and descriptions that are written do not reflect the state of mind of the person. In this case, a different means of measuring the internal state of being of the person is required and we are touching on the subject of **ethics** and **privacy**. How far are you willing to go to find out details of the personal life of a developer? The ethic answer would be: nowhere.

V. Developed solution

During this work, we developed a *Python* application which gives a quick overview of a *repository*'s *Pull Requests* and the selected *Pull Requests*'s emotional classification to help project managers decide on which *Request* they should be more attentive to detail. As explained in the introduction, every *commit* and *comment* of the author should be analyzed to quantify the quality of the modifications.

The application connects to a GitHub account via a *Username* and an *OAuth token* (GitHub authentication token), and shows the repositories of that account. A background service polls the repository at a defined interval and checks to see if reactions to *Pull Requests* have been submitted. In the case of a new *Pull Requests*, the application tries to quantify the quality of the modifications of the program by extracting emotions.

VI. Emotion Extraction

Emotions can be either positive or negative and are expressed within *commit* descriptions or comments by the developers or reviewers of that project. For example: those *emoticons* can be ”thumbs up”, ”thumbs down”, ”laugh”, confused” of ”heart”. We affected a ”emotional value” to each possible reaction and our application sums up all those values to get a balanced result, which can be either positive or negative. Based on that result, we choose to attribute a specific label to that *Pull Request* when the balanced result if negative.
The main advantage of labeling the *Pull Request* is to help maintainers and project managers able to determine with just a glace, where a more thorough review of the request is likely to be required. This can be used by *Human Resources* to try and help the developer that is in a bad emotional state. It can also be used as a way of identifying clean code more easily.

At time of writing of this article, we do not use machine learning or any other tool to parse the source code comments nor any textual content and try to extract the emotions of the developer itself. We also do not look at the code itself.

VII. Results

Using binary balanced results of emotions leads to binary labels, which leads to a very simple way of classifying *Pull Requests* after they have been submitted.

A *Pull Request* is classified as *Attention Required* or *Clean*. This classification allows managers to devote additional resources to the testing of problematic Pull Requests by informing the code reviewer and/or testing team that the developer may have been in a negative emotional state. This introduces some questions that the manager should think about:
Which pull request should be accepted ? What should be done with pull requests that have not been qualified as positive by other developers ? What should be reviewed and how ?

The results on a real repository like Itseez' OpenCV project[1] which has at time of the test runs 45 open Pull Requests and 6256 closed Pull Requests that have been approved[2] has shown exactly what we suspect: **the classification is completely useless without context and at least a form of code analysis**.

VIII. Implementation and tests

This application has been developed as a *Python* set of files. There is a main file which instantiates the different classes used in that project, that are bundled in a periodic service used for polling at a regular time basis.

We implemented a *continuous integration* and a *continuous delivery pipeline* in order to build a robust and (hopefully) bug-free program.

We implemented unit tests on the different *Python* classes and integration tests on the whole project.

We use *Travis CI* as well as *Jenkins* in a *Docker* environment to demonstrate this pipeline.

IX. Improvements proposal

In that project, we used a simple balanced sum of emotions given by code reviewers and other developers on the Github page concerning a pull request. That could be greatly improved by implementing other metrics to estimate the quality of the pull request.

The first thing that could be implemented is the Open Affect API which could help us to link emotional measurement with the associated pull request, which could be more precise than getting emotions manually set by reviewers.

X. Conclusion

It this project we successfully implemented an application which tracks all pull request made on a GitHub repository, in order to measure the quality of a pull request. This has been done by using feedback given by other developers and this can lead to project management tasks such as validating or rejecting some pull requests.

This project could be enhanced by using new metrics as quality measurements as explained in previous section, but it is already working and can set labels depending on the reviews.

This project has been a good opportunity to implement in *continuous integration* and establishing a *continuous delivery pipeline*, including *unit* and *integration* tests.

[1] https://github.com/opencv/opencv

[2] https://github.com/opencv/opencv/pulls?utf8=%E2%9C%93&q=is%3Apr%20is%3Aclosed%20is%3Amerged

ACKNOWLEDGMENT

The authors would like to thank the GitHub developer team for providing such a powerfull API to access repositories and their data.

REFERENCES

[1] Openaffect API, https://github.com/openaffect

[2] Github API, https://developer.github.com/v3/

[3] GraphQL, https://developer.github.com/v4/

[4] Y. Yu, H. Wang, G. Yin, and C. X. Ling. Reviewer recommender of pull-requests in github. In Proceedings of the IEEE Conference on Software Maintenance and Evolution, pages 609612, 2014.

[5] Y. Yu, H. Wang, G. Yin, and C. X. Ling. Who should review this pull-request: Reviewer recommendation to expedite crowd collaboration. 21st Asia-Pacific Software Engineering Conference, pages 335342, 2014.