CSC 510 Software Engineering Project 1

Setu Kumar Basak
Conor Thomason
Keertana Vellanki
Muntasir Hoq
Matthew Sohacki
sbasak4@ncsu.edu
cjthom24@ncsu.edu
kvellan@ncsu.edu
mhoq@ncsu.edu
mjsohack@ncsu.edu

Abstract

We made every effort to follow the Linux Kernel practices to the best of our abilities while working on the project. This paper provides an overview of the connection between Kernel practices and how we worked around them for our project:

1 Introduction

Sentiment analysis is among the most rapidly increasing research areas in computer science. It is usually quite difficult to keep up with all of the development taking place in the area. By analyzing the data provided in different methods, we expect to achieve our goal of accurately predicting a user's sentiment in our project. Although it is still in its early phases of development, this project has the potential for application to a variety of sectors that could be beneficial to society.

2 Zero internal boundaries

Throughout the project, our team has maintained open lines of contact. Any important project decision was thoroughly discussed in advance. We actively discussed and decided the roles of each team member throughout each meeting. We met each other in person weekly and also held zoom meetings whenever needed. We also maintained a discord group for our project where we communicate with each other regarding any issues and changes made by anyone in any part of the code. Suggestions and changes, from all the team members, were always welcomed and encouraged. We collaborated to update the GitHub repository, and everyone had the opportunity to edit, add, and remove any of the content. Maintaining the version control system led us to have zero internal boundaries and pushing to the main branch on a regular basis helped us to understand and gain access to any change or any newly added feature in the system.

3 No regressions rule

The anticipated changes or updates are unlikely to be deleted in the near future. Test cases were used to ensure that existing functionality persisted in new releases. Functionality that was not covered by test cases and regression were immediately made the top priority; new functionality was not implemented until the problems were resolved. We made sure that everyone worked in a specific setting. Everyone updated the requirement.txt file regularly which helped others to get aligned with the newly updated version and setting of the project. Though we used different languages to develop, keeping our requirements and environment in sync with each other let everyone work without any clash or conflict, and whenever any upgrade was needed the system did not break down.

4 Consensus-oriented model

We held weekly team meetings to discuss overall progress on the project and identify any new issues or tasks that should be worked on. We used the team's active chat channel on Discord to discuss and resolve any pending issues. All project-related decisions are made by the consensus of the group. While working on the project, the CODEOFCONDUCT.md has strict guidelines that were followed. For new users, these guidelines are a nice place to start. The GitHub repository's CONTRIBUTING.md file has been updated with information on how users can assist improve the project.

5 Distributed development model

For the project, our team worked as a united group. All team members were able to attend and participate in the project because the meetings were scheduled at times that were convenient for them. At the conclusion of the meeting, we had settled on the dates and venue of the next meeting. We noted down every decision taken in the meetings and any issue that we faced so that we worked synchronously. We maintained Github issues and the Linear app to divide the whole project into small parts to work on and regularly commit to the Github repository so that everyone can have access to the update. We always supervised other members' work so that everyone can have an idea of the whole project and there is no conflict due to any upgrade or new addition of features. In order to tackle problems, we collaborated as

a group. Our project has a DOI Badge and a License. Our project website is now up and running.

6 Short release cycles

Long release cycles mean integrating codes after a long time. In our case, committing after a long time and integrating them may result in conflicts and low-quality integration. Committing in short cycles reduces this chance and paves the way to add more fundamental features. Short release cycles help with integrating codes on a regular basis which helps minimize conflicts. This also helps other developers to get the chance to look into any changes made in any part of the system. All the commits we have made until now are going into the main branch. The main branch is being deployed to the production CI/CD pipeline for every commit.