Connection between the Project-1 Rubric and Linux Kernal Best Practices
Group 22 - Donatify

Akash Gupta
North Carolina State University
Raleigh, NC, USA
agupta57@ncsu.edu

Nagaraj Madamshetti
North Carolina State University
Raleigh, NC, USA
nmadams@ncsu.edu

Arun Srinivasan P
North Carolina State University
Raleigh, NC, USA
apartha4@ncsu.edu

Nitesh Mishra
North Carolina State University
Raleigh, NC, USA
nmishra4@ncsu.edu

Sumit Singh
North Carolina State University
Raleigh, NC, USA
ssingh57@ncsu.edu

ABSTRACT

In the following paper, we describe the links between the project-1 rubric items and the Linux Kernel Development Best Practices. Our project Donatify, is a donation hub that allows for easy and flawless donations from one person to another. Multiple users sign up and post products/items which are then notified to those users who have an interest in such items.

KEYWORDS

Linux Kernel Best Practices, Software Engineering

1 INTRODUCTION

The Linux kernel is one of the most dynamic open source projects which has multiple stable releases every few months. Thinking about the sheer amount of code makes us wonder how the Linux community has kept this project well alive. These are the 6 Kernel Development best practices which has allowed this project to flourish continuously.

(1) Short Release Cycles
(2) Distributed Development Model
(3) Consensus Oriented Model
(4) Tools Matter
(5) The No-Regressions Rule
(6) Zero Internal Boundaries

2 KERNEL DEVELOPMENT BEST PRACTICES

Given below is the connection between the above mentioned Kernel Practices and the Project-1 Rubric.

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

2.1 Short Release Cycles

Short release cycles were introduced to curb the issues due to the integration of large chunks of code, which would put pressure on the developers to deploy an unready or unstable release. Majority of the time, this would lead to delays and dissatisfaction among the customers.

Continuously integrating newer features into the code base in shorter release cycles would allow for developers to scrutinize their code in a more calm manner. At the same time, the customers would also be getting a new feature in shorter time cycles.

The rubric mentions the use of shorter release cycles for the above mentioned reasons. Having very few commits might lead to the final product, but this would come a risk of having longer development cycles and integration. This could easily lead to multiple glitches and bugs.

In our project, we have made multiple commits across time, with each of them having very small functional changes or changes pertaining to documentation, thereby ensuring short release cycles.

2.2 Distributed Development Model

A single person working on the entire project, while others just keep looking at the code would never lead to a good model.Having different ideas and building upon each of them is what makes a model more balanced.

Allowing one person to handle the entire workload would be very cumbersome and could in-turn destroy the complete project. Allowing different individuals to work on their area of familiarity would enable the seamless integration of code without breaking any existing architecture. This Distributed Development Workflow also does not compromise on the stability of project.

The rubric mentions that the workload should be distributed equally among the team members. In our team, Akash and Nagaraj handled the frontend modules while the backend API development was done by Arun, Nitesh and Sumit. The documentations were

handled on an equal distribution from each member of the team. This allowed each member of the team to work on the part which they had more experience with and lead to a stable release before the deadline.

The rubric also mentioned about the number of commits made by the developers. This is also a good factor to ensure each member is contributing towards the project. Commit messages were also kept short and simple so that each member of the team could understand the changes made without any issues.

2.3 Consensus Object Oriented Model

The Linux Community works on the ethic that every code change should be integrated only after all the respected developers accept the proposal. If even one of the respected developers is against the change it has to be reviewed with the correct changes. This also allows each individual of the team to have a common goal and mindset of what is expected.

The rubric mentions the use of CONTRIBUTING.md which specifies how anyone can contribute to extending our project Donatify. CONTRIBUTING.md tells the developers the particular coding standards which need to be followed while integrating their changes to the codebase. Discussion via pull requests, issues, chat channels, google meets were other alternatives via which the team coordinated on their respective changes.

2.4 Tools Matter

The use of tools has increased over time and surely helps in the coordination of the code development in a better manner.

In our project, we have made use Git as a version control system. This allows for each member of the group to always have the latest version of the code. Git helps the developers to make any kind of change without fear of conflict from other members. Opening multiple Pull Requests and branches also helped for each individual to work on separate functionalities. Other automated tools include Zenodo for Citation, pdoc3 and jsdoc for Documentation, and codecov for code analysis.

2.5 The No-Regressions Rule

Every single change should be monitored once implemented to ensure that there are no regressions present. The Linux community follows the rule that if a given kernel works in a specific setting, all subsequent kernels must also work in the same setting.

The rubric mentions the use of GitHub Actions that includes, but not limited to, test cases, syntax checkers, code coverage, and automated analysis, which allows the project to be routinely checked ensuring no regressions.

2.6 Zero Internal Boundaries

In general, groups have members working in their specific specialities. However, it is better for each developer to also work on other domains as long as the changes are accepted by the respected developers of that particular domain. Its always better for each individual of the team to have broader sense of the project.

In the rubric, it is mentioned that each individual should understand each and every part of the code. It also asks if each member of the group can access every tool in the project, and can edit every single project file. We had multiple group calls and discussions to gain insights of the different modules of the project, which were developed by different individuals.

3 REFERENCES

(1) https://medium.com/@Packt_Pub/linux-kernel-development-best-practices-11c1474704d6