Reflection on Project 1 Rubric and Linux Kernel Best Practices

Sneha Aradhey
North Carolina State University
Raleigh, North Carolina, USA
svaradhe@ncsu.edu

Suraj Mallikarjuna Devatha
North Carolina State University
Raleigh, North Carolina, USA
sdevath@ncsu.edu

Shreyansh Prajapati
North Carolina State University
Raleigh, North Carolina, USA
saprajap@ncsu.edu

Himanshu Gupta
North Carolina State University
Raleigh, North Carolina, USA
hgupta6@ncsu.edu

Ashwini Nayak
North Carolina State University
Raleigh, North Carolina, USA
aunayak@ncsu.edu

Figure 1: Image from The Linux Kernel Report, 2017

ABSTRACT

This paper aims to reflect upon the Project 1 Grading Rubric and compare it to Linux Kernel best practices. We map the various rubric criteria from Project 1 and present descriptions of each practice. We implemented these practices as we worked on our team project and in this paper we describe how in more detail. Overall, we found these practices greatly beneficial and were able to understand it throughly through the hands-on implementation of our project.

KEYWORDS

Linux, best practices, development, application, GitHub

1 INTRODUCTION

The Linux Kernel Report (2017) provides numerous instances of what contributes to best practices for project development [1]. Throughout this paper, we henceforth refer to these practices as "Linux Best Practices". The graduate curriculum for software engineering at North Carolina State University (NCSU) comprising this Project 1 rubric is on par with these ideals. Our objective is to compare the rubric with Linux Best Practices and present the mapping with detailed examples from our project "WolfTrack"(Team 36 group project)[2].

2 COMMENTS ON LINUX BEST PRACTICES

The below section comprises of detailed descriptions of Linux Best Practices and how our project "WolfTrack" aligns with the same. This document will also explain the major examples that showcase these practices implemented.

2.1 Short Release Cycles

Considering the entire duration of Project 1 was relatively short, having short release cycles was a vital component that contributed to the success of the project. It enabled us to pin-point issues quicker and hence resolve them faster as we were tackling them over shorter time duration. As the duration was just a month, this had a huge impact as it restricted the amount of code integration as stated in Linux Best Practice.[1] Additionally, having frequent commits by all team members made it possible to keep the development process as up to date as possible thus acting as a catalyst to progress. This is because it reduces overhead which could otherwise accumulate.

We created separate branches for each feature and was then merged to the production branch after another member reviewed it. As the defined features were not too bulky in terms of code, it supported team members in fixing bugs and presenting new perspectives of the feature.

2.2 Requirement for a Distributed Development Model

The scalability of the project depended upon having a distributed development model. This meant that the project work and responsibility was evenly distributed across the project. Had the project responsibility fallen on a single developer, as the project scaled up, the developer would find it difficult to keep up with the project pace. Therefore it is crucial to have a team to cover all grounds of the responsibility. The Project 1 rubric supports this practice by including a section on how each individual user contributed to the repository evidenced by GitHub commits. Additionally, it also

mentions utilization of issues to ensure work can be easily defined and distributed among the team members.

From a "Wolftrack" perspective, all the members in the team used GitHub and worked in parallel to create the various features of the project. We found GitHub to be perfect with respect to this as it offers bountiful features for distributed development. This included the creation of multiple branches for different components which later merged to main after successful testing.

2.3 Tools matter

Tools are crucial for development projects as it works as glue by holding it up. It is emphasised in Linux Best Practices that without tools, projects the size of Linux would not be successful [1]. The Project 1 rubric covers this in more detail as it sheds light upon the various version control tools, style checkers, syntax checkers as well as other automated tools. Not only do these tools provide additional functionality, but also supports better team coordination, communication and code standards.

"WolfTrack" is in line with this. In our project, we utilized the following two code formatters:

- `Prettier - Code Formatter`: Prettier is an opinionated code formatter. It enforces a consistent style by parsing your code and re-printing it with its own rules that take the maximum line length into account, wrapping code when necessary.
- `JSS-CSS-HTML Code Formatter`: This extension wraps jsbeautify to format, prettify and beautify JS, CSS, HTML, JSON file by using shortcuts, context menu or CLI.

We used the code checker:

- `Pylint`:Pylint is a source-code, bug and quality checker for the Python programming language.

Additional tools used included:

- `Miro`: Miro is a scalable, secure, cross-device and enterprise-ready team collaboration whiteboard for distributed teams.
- `Discord`: Discord is a VoIP, instant messaging and digital distribution platform.

The entire team collaborated on the same tools to ensure optimal productivity and uniformity.

2.4 Importance of a Consensus oriented model

Maintaining a consensus-oriented model ensures that all team members of the project are on the same page and in agreement with the updates that are made and are to be made. A change can not be merged and incorporated in the development model if a team member with the responsibility is not in aggreement with it. This is in line with "proposed change will not be merged if a respected developer is opposed to it" and " no particular user community is able to make changes at the expense of other groups" from Linux Best Practices [1].

Team developers of "WolfTrack" created a Discord server to aid with this collaboration. Comments, discussions and updates were communicated via the application supporting "Issues are discussed before they are closed" and "Chat channel: exists". This paved the way for seamless communication between teammates to brainstorm for features, address concerns and speak out about problems. Discord proved to be helpful to apprise the team of project status and updates. Comments were added to issues, commits and pull requests in GitHub to gain consensus. Team members reviewed each others work before pulling or merging requests.

2.5 "No Regression" rule followed

This rule refers to the ability of a project to endure updates without breaking older versions or previous functionality [1]. Though Project 1 had a short lifespan limiting potential regression concerns, the rubric requirements with respect to documentation and testing cover this aspect. Using workflows for the old and new functionalities can help mitigate future regression and good documentation provides a good understanding of older versions of the project. GitHub actions proved to be a great tool for handling test cases and automatically checking for failed test cases upon pull requests. Our team took extra care to follow the above practices to ensure no regression was occurring. Good documentation helped us with this as well.

2.6 No internal boundaries in the project

Having no or zero internal boundaries is beneficial to the team. This is because boundaries could cause hindarance and restrict a team members involvement to different parts of the code base. By limiting editing rights or communication, that causes a strain in communication and collaboration between different team members. Everyone responsible towards the project or sub task must not be prevented to work on it in any form. The rubric phrases it as "workload is spread over the whole team" and rows containing evidence that all team members can run code and configure tools. "WolfTrack" achieved this practice by using tools like Miro to brainstorm ideas and allowing every member to share their thoughts. The agreed upon features was then broken into sub tasks and put in the pipeline for development. Every member of the team also agreed upon the tools that would be used to make working on the project fair game play. The discord channel was highly active with progress updates and other communication which provided the team to work with zero internal boundaries.

3 CONCLUSION

The Linux Best Practices aligns perfectly with the Project 1 rubric. Exploring these practices was highly beneficial to the team as a whole. Developing the project, bearing these practices in mind, helped us understand the importance of these practices and why we were graded in this manner. We would definitely use and recommend these practices in our future projects.