Comments on Project 1 Rubric and Linux Kernel Best Practices

Emily Tracey
North Carolina State University
Raleigh, North Carolina, USA
etracey@ncsu.edu

Leila Moran
North Carolina State University
Raleigh, North Carolina, USA
lmmoran@ncsu.edu

Peeyush Taneja
North Carolina State University
Raleigh, North Carolina, USA

Jonathan Nguyen
North Carolina State University
Raleigh, North Carolina, USA
jhnguye4@ncsu.edu

Shraddha Mishra
North Carolina State University
Raleigh, North Carolina, USA
smishra9@ncsu.edu

Figure 1: Screenshot from The Linux Kernel Report, 2017.

ABSTRACT

This paper aims to connect Linux Kernel best practices and the rubric for Project 1. We present descriptions of each practice and how different rubric criteria from project 1 relate to each practice. We applied those practices in our team project and explain how through this paper. Ultimately, we found we were able to connect the rubric to all of the Linux Kernel best practices and our team benefited greatly from these practices.

KEYWORDS

Linux Best Practices, GitHub, development model

ACM Reference Format:
Emily Tracey, Leila Moran, Peeyush Taneja, Jonathan Nguyen, and Shraddha Mishra. 2021. Comments on Project 1 Rubric and Linux Kernel Best Practices. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The Linux Kernel Report from 2017 gives a list of some best practices for project development [1]. Throughout this paper we will refer to these as "Linux Best Practices." The Project 1 rubric for North Carolina State University's graduate software engineering class incorporates these ideas. Our goal is to comment on how this rubric aligns with Linux Best Practices and provide some concrete examples in our Team 25 group project.

2 COMMENTS ON BEST PRACTICES

The following sections will highlight each best practice and how Project 1 aims to follow each practice. Each section will also include examples from our team's project.

2.1 Short release cycles are important

While Project 1 was a relatively short project, the short release cycles practice still holds. Even just the due date for the project not being a semester long prevents huge amounts of code integration like Linux best practice states [1]. Additionally, the project rubric was able to address this in the last point confirming that project members are committing often. Having frequent commits from team members allows local versions of the project to stay up to date and prevents overhead with integration.

Our team generally followed the practice of committing whenever functionality of a new feature was completed or whenever a team member requested it. Waiting for a complete feature limited "broken" code that was in existence and kept the number of commits up since our defined features were relatively small and

accomplished in a matter of days. The second case was used widely to aid members in fixing bugs or adding a new viewpoint.

2.2 Process scalability requires a distributed development model

This practice requires that responsibility across a project to be distributed. A single talented developer can be easily outmatched as a code base grows [1]. Project 1 rubric ensures this by including points about how all users are contributing to the repository with some evidence existing in commits. The rubric also requires the use of issues which allows work to be easily defined and divided among group members.

GitHub provides functionality to view contributions by each user, so we met the rubric requirements by ensuring members were contributing and there contributions are noted within the repository. We also utilized GitHub's issue tracking and progress boards distribute development.

2.3 Tools matter

The Linux Best Practices emphasizes that tools are important and that a project the size of Linux would collapse without appropriate tools holding it up [1]. We see evidence of this in the project 1 rubric in several rows including: version control tools, style checkers, syntax checkers, and other automated tools. These tools do not only allow for additional functionality for a project, but also enables better team collaboration and code standards. Keeping a well maintained code base is a desire for all developers, so hundreds of tools have been created to aid in this goal. Having it emphasized by in the project rubric simply fits the way development happens today. Developers need to make use of those hundreds of tools out there.

Our project used tools such as package managers like Maven [2] and npm [4] to manage our Java and React dependencies respectively. We found the intelliJ IDE extremely helpful in managing our coding syntax and style with the addition of ESLint [3] to mange code formatting for the frontend code. These are just some of the many tools used.

2.4 Consensus-oriented model is important

A consensus-oriented model ensures that members of project agree on what is being added and changed. Linux Best Practice emphasizes "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" [1]. The rubric handles this component by "issues are discussed before they are closed" and "Chat channel: exists." Both of these points require proof of team communication and allow opportunities for group members to speak out about a problems or concerns they might have.

Our team used a Discord channel to discuss project updates and concerns. We also used our channel to discuss GitHub issues. When appropriate, we attached comments to GitHub issues and pull requests to gain consensus as well. Finally, we utilized pull requests that were open and closed by separate team members frequently to prevent a single member contributing non agreed upon code.

2.5 Strong "no regressions" rule

Linux Best Practice "no regressions" rule refers to the ability for a project to update without breaking older versions or previously existing functionality [1]. While Project 1 has a short time scale and the opportunities to regress are few, the rubric requirements concerning documentation and testing cover this practice. Using workflows for all functionality (old and new) can help mitigate regressions for the future and extensive documentation allows for a snapshot of older versions of the project. Having these snapshots and reasoning can help future designs not regress the old.

GitHub actions provided a great tool for managing tests and automatically checking for failed tests upon pull requests. We used several workflows to manage aspects of the system to confirm new changes are not regressing any aspect. Documentation especially in the form of JavaDocs helped us meet this practice as well.

2.6 There should be no internal boundaries within the project

Having zero internal boundaries is beneficial to all team members. Boundaries that could exist include restricted access to parts of the code base, limits of editing rights, or even communication. Team members should be free to work and update any portion of the project provided they have a valid reason to. This practice is also demonstrated in the rubric in "workload is spread over the whole team" and the rows containing evidence that all team members can run the code and configure tools across the code base. Keeping members aware of all parts of the system is the first step in erasing boundaries.

We accomplished this practice by not only allowing members to choose which features they wished to develop, but also by allowing members to switch or adapt as needed. Tools that are used were first confirmed by the team and then implemented by anyone. Specifically, the package.json (frontend configurations) and pom.xml (backend configurations) were accessible to the whole team and were updated by several members. To avoid communication barriers and confusion on where people were working in the code base we used Discord chat channel and regularly checked GitHub.

3 FINAL THOUGHTS

The Linux Best Practices connects heavily to the project 1 rubric and exploring these connections has been beneficial for our team to understand how and why we are graded the way we are. We will definitely be considering the Linux Best Practices in our future development not only for this semester, but in our professional careers as well.

REFERENCES

[1] 2017. State of Linux Kernel Development 2017. The Linux Foundation®, 25–26. https://www.linuxfoundation.org/resources/publications/state-of-linux-kernel-development-2017/
[2] 2019. *Apache Maven.* https://maven.apache.org/index.html
[3] 2021. *ESLint.* https://eslint.org/
[4] 2021. *npmjs.* https://www.npmjs.com/