Linux Kernel Best Practices and Project Rubric Connection

Damilola Babalola
djbabalo@ncsu.edu
North Carolina State University
Raleigh, NC, USA

Aaron Dias Barreto
aadiasba@ncsu.edu
North Carolina State University
Raleigh, NC, USA

Swarangi Gaurkar
sgaurka@ncsu.edu
North Carolina State University
Raleigh, NC, USA

Leo Hsiang
yhsiang@ncsu.edu
North Carolina State University
Raleigh, NC, USA

Kritika Javali
ksjavali@ncsu.edu
North Carolina State University
Raleigh, NC, USA

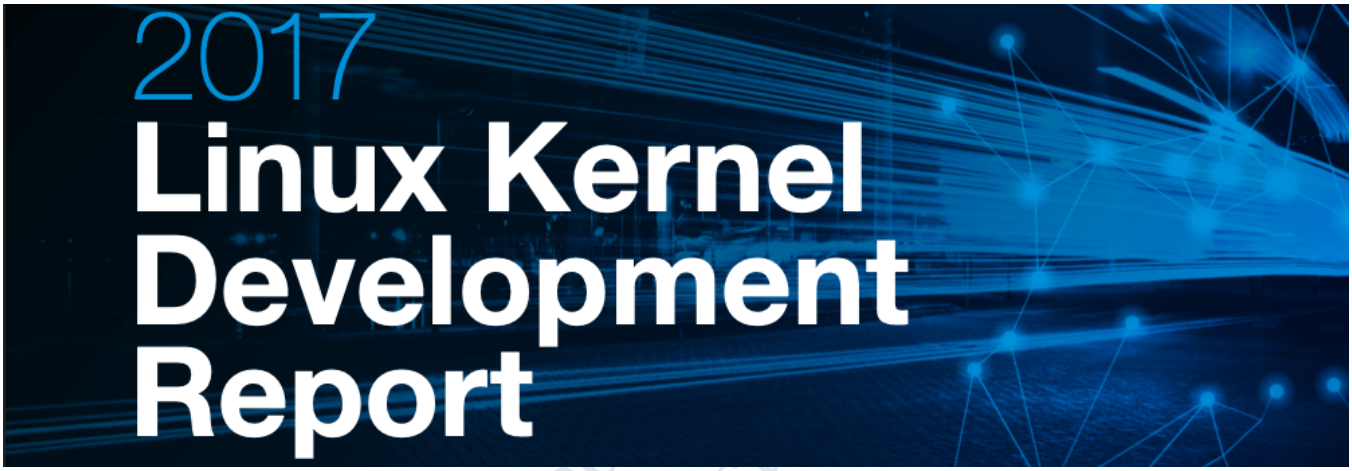

Figure 1: Screen Shot from the 2017 Linux Kernel Development Report

ABSTRACT

This paper states the connection between the Linux Kernel Best Practices and the Project Rubric provided as part of the Software Engineering coursework. We present descriptions of each practice and how different rubric criteria from project 1 relate to each practice.

KEYWORDS

linux kernel, project rubric, best practices

ACM Reference Format:
Damilola Babalola, Aaron Dias Barreto, Swarangi Gaurkar, Leo Hsiang, and Kritika Javali. 2022. Linux Kernel Best Practices and Project Rubric Connection. In *Proceedings of Software Engineering - CSC 510 Fall 2022 (Conference acronym 'SE).* ACM, New York, NY, USA, 2 pages. https://doi.org/nnnnnnn,nnnnnnn

Conference acronym 'SE, Oct 07, 2022, Raleigh, NC

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

1 INTRODUCTION

The Linux Kernel Best Practices contains the best lessons by the kernel project team with 26 years of experience on developing and actively maintaining the Linux Kernel project. Linux Kernel is one of the largest and widely used open source project, so we will try to understand the lessons behind the success and apply the best practices to our own project.
The six main best software development practices attribute to the Linux Kernel's success are:

- Short Release Cycles
- Distributed Development Models
- Tools Matter
- Consensus-Oriented Model
- No Regression Rule
- Zero Internal Boundaries

The Project Rubric provided for CSC510 Software Engineering coursework is a detailed rubric provided by the teaching staff to assess if a particular repository is a good open source repository or if the development process meets all the expected standards.
These rubric metrics also include the Software Sustainability Evaluation questionnaire that provides good opportunity to review the main issues that affect the sustainability of our project. Using these rubric metrics, we can ensure that the project is designed and developed in such a way that it has a better reach to the users and

developers of the open source project.
In this paper we will try to list down the connection between the main 6 Linux Kernel best practices and the Project Rubric.

2 COMMON METRICS

2.1 Short Release Cycles

In the past, Linux kernel development used to do major releases once in a few years. This gave rise to a lot of hindrances in the release. Not only extensive integration at once required a lot of attention from developers, but also considerable delays in getting new features to users, which was frustrating to users and distributors alike[1]. Shorter release cycles mainly enable teams to get critical functionality out sooner to solve customer problems. Having short releases cycles can help the development process in lot of other ways.

- Release a lot of features to the user quickly
- Prioritize tasks based on the release dates.

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.

2.2 Distributed Development Models

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]. Having distributed development models can help the development process in many ways.

- More evenly workload and responsibilities among team members
- Deal with a large amount of changes without sacrificing review or quality

The project rubric was able to address this metric by specifies "Workload is spread across the whole team" and "a track record that everyone teammates is contributing a lot".

2.3 Tools Matter

The Linux Best Practices emphasizes that tools are important and that a project with the size of Linux would collapse under its own weight without using appropriate tools [1]. Keeping a well maintained code base is a desire for all developers, so hundreds of tools have been created to aid in this goal. Using the right tools can help the development process in lot of other ways.

- Lesser things to integrate or worry about
- Save development time

We see evidence of this in the project 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.

2.4 Consensus-Oriented Model

A consensus-oriented model ensures that most members of project agree on what is being added and changed. Linux Best Practice emphasizes "No particular user community is able to make changes at the expense of other groups" [1]. Applying consensus-oriented model can help the development process in lot of other ways.

- The software might be more suitable for larger group of users
- Better moral and adherence to decisions

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.

2.5 No Regression 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]. Applying no regression rule can help the development process in lot of other ways.

- Stabilize the software
- Detect potential regression bug early on

The rubric requirements concerning documentation and testing cover this practice. We checks if our tests are in place with good code coverage to ensure that everything works as expected. We used several workflows to automated checks different aspects of the system to confirm new changes are not regressing any aspect.

2.6 Zero Internal Boundaries

Kernel developers generally work on specific parts of the kernel; however, any developers can make a change to any part of the kernel as long as the change can be justified [1]. Having zero internal boundaries makes sense in development process because team members should be free to work and update any portion of the project provided they have a valid reason to. Having zero internal boundaries can help the development process in lot of other ways.

- Lesser things to integrate
- Fix bugs from the last release cycle soon

This practice is demonstrated in the rubric with containing "evidence that all team members are working across multiple places in the code base". Making sure team members aware of all parts of the system is great step in erasing boundaries.

3 CONCLUSION

The Linux Best Practices connects heavily to the project rubric and exploring these connections has been beneficial for our team to understand the rationale behind why we need to follow the best practices and project rubric. We will definitely be considering the Linux Best Practices in our future development not only for this semester, but in our professional careers in software engineering as well.

REFERENCES

[1] 2017. State of Linux Kernel Development 2017. The Linux Foundation®, 25–26. https://www.linuxfoundation.org/resources/publications/state-of-linuxkernel-development-2017/