LectureAid Rubric Comments

Manoj Kumar
mtkumar@ncsu.edu
North Carolina State University

Rakesh Muppala
rmuppal@ncsu.edu
North Carolina State University

Renji Joseph Sabu
rsabu@ncsu.edu
North Carolina State University

Sayali Parab
snparab@ncsu.edu
North Carolina State University

Ashwin Das
adas9@ncsu.edu
North Carolina State University

Ashley King
anking4@ncsu.edu
North Carolina State University

ABSTRACT

The Linux Kernel Best Practices have been created from lessons learnt over 26 years of development experience. Here in this document we go through each principle of the Linux Kernel Best Practice, how it ties into the Project 1 Grading Rubric, and how we have emulated these practices while developing the code for our project.

CCS CONCEPTS

• **Software Engineering**; • **Best Practices** → *Linux Kernel Development*;

KEYWORDS

Software Engineering, Linux Kernel Best Practices, LectureAid

1 INTRODUCTION

The Linux Kernel is a great example of open source collaborative development and constructive collaboration. Despite the large number of contributors and length of the project the Linux Kernel has grown in terms of success ever since its conception in 1991. To avoid development chaos the Linux project shares some key principles that we incorporated as part of our project guidelines [2]. These guidelines are listed below:

- Short Release Cycles
- Zero Internal Boundaries
- The No-Regressions Rule
- Consensus Oriented Model
- Distributed Development

These 5 rules are adhered to by the Linux open source community, and have been vital to their success over the last 30 years. In this paper we will discuss theses principles, and compare and relate them to the grading criteria part of our Project Rubric 1[1]. We will also examine how we applied these principles in our software development to ensure efficiency and good collaboration.

2 LINUX KERNEL BEST PRACTICES

2.1 Zero Internal Boundaries

One of the linux kernel best practices is Zero Internal Boundaries. This practice refers to the fact that all the developers working on a project should have access to all the tools used in the development of the software. They should be able to run all the pieces of software on their work environment, and should also be able to work on any parts of the different software pieces as they see necessary.

The Project1 Rubric checks to see if we have achieved this best practice by grading us on the following criteria:

- Evidence that the whole team is using the same tools: everyone can get to all tools and files
- Evidence that the whole team is using the same tools (e.g. config files in the repo, updated by lots of different people)
- Evidence that the whole team is using the same tools (e.g. tutor can ask anyone to share screen, they demonstrate the system running on their computer)
- Evidence that the members of the team are working across multiple places in the code base

We have achieved this by doing the following. Everyone has been using the same Python version and IDE PyCharm for coding purposes. We have ensured that the virtual environments being used in Pycharm for development have the same libraries installed, so that everyone can pull the entire code repo and run all the scripts. The requirements.txt file helped ensure that everyone had access to the same libraries and models. Everyone's code is being checked with PyLint when committing to the repo, to ensure the formatting and health of the code is good. Although initially, each member of the team worked on different steps/files in the project workflow, after an initial prototype was created, each member of the team had to work on and commit to different scripts/files. Test cases were distributed, and this helped ensure that members of the team were working across multiple places in the code base.

2.2 No Regressions Rule

The "No regressions rule" in Linux Kernel's best practices states that if a given kernel works in a specific setting, then all the subsequent kernels must work there too. This will give some assurance to users that any upgrade on the software will not break their systems, so that the users will follow the software as it updates.

The following checks in the rubric matches with the No Regressions Rule:

- Test cases exist
- Test cases are routinely executed
- The files CONTRIBUTING.md lists coding standards and lots of tips on how to extend the system without screwing things up

In our project, the test cases that we have written makes sure that new code written will have to pass existing test cases as well in addition to the one written for itself. Since the tests are routinely executed, we make sure of that fact. When implementing a new feature we create and work on a separate branch. And when that branch is merged back to main, TRAVIS CI will run to check all the tests, both the old and new, so its easy to check and confirm that the existing functionality was not affected. We create new issues, in the event that any problems do occur to existing functionality, to make

sure that we address the problem promptly so as to conform to the No Regressions rule. And finally, the new people who contribute the project will have to follow CONTRIBUTING.md which makes sure that they have enough information to contribute to the existing project without breaking existing functionality.

2.3 Consensus-Oriented Model

The "Consensus-Oriented Model" best practice states that that when developing the solution, everyone should have a say in the development, and changes are only made when the group comes to a consensus. This ensures that all team members are able to have a say in every code change, and their thoughts will not be ignored. This is a large part of open source development, ensuring all users have a chance to contribute equally.

The Project1 Rubric checks to see if we have achieved this best practice by grading us on the following criteria:

- Issues are discussed before they are closed.
- Is your source code stored in a repository under revision control?
- Are e-mails to your support e-mail address received by more than one person?

We have achieved this by doing the following. Discussing issues before closing them requires the entire team/development teams input before and changes are made. We achieved this by leaving comments on every merge request, or by discussing over discord. Examples of comments included "Reviewed" or "Looks good" to say that the user had reviewed the branch and it looked good to merge. An example of a comment to say the branch did not look good is: "Closing PR as errors not fixed with pylint config. Additional research required". By making other team members okay a merge request, this ensures that any changes made are only made when there is a group consensus. Having the source code stored in a repository under revision code allows all team members to have access to the same code, allowing all team members to contribute as a consensus. Everyone being able to read emails ensures that the emails are not being held secret by one person, instead shared as a group. By following a Consensus-Oriented model, we were able to ensure all team members are able to contribute equally and have their voice heard equally.

2.4 Distributed Development Model

Distributed Development Model is a software development model in which a team spread across different physical locations collaborates and communicates with each other using internet based platforms to build software.

The Project1 Rubric checks to see if we have achieved this best practice by grading us on the following criteria:

- Issues are discussed before they are closed
- Chat channel: exists

We have achieved this by doing the following:

- We split the workload across the entire team and record the contribution made by each member through the number of commits on the GitHub statistics page.
- We had conducted online meetings over Google Meet where we discussed the work done over the past week and decided and split what each person has to do over the coming week.
- We had a discord group dedicated to project discussion where people frequently discussed what they did, their pushed changes, work progress and about any issues any or multiple team members were facing.

2.5 Short Release Cycles

Linux kernel development community switched to short release cycles, which addressed all the issues with long release cycles. So now, new code was immediately pushed and integrated into a stable release. Plus, continually integrating new code allows for introducing fundamental changes into the code and forming a stable base without causing major disruptions.

One of the project rubrics,

- Use of version control tools, Number of commits: by different people, Use of git and Git.
- Use of branches, and commits. This further confirms that the commits have been merged by the developers through branches.

This assures that the entire team is reviewing the work and then it's pushed to the main branch resulting in a stable base framework. But still, this being the first stage of the project, its rare to have released since its the Beta version of the the project which includes the core base for future road-map.

3 CONCLUSION AND FUTURE WORK

Following the Linux Best Practices helped ensure our software adhered to open-source development best practices. By ensuring there were no internal boundaries, all team members were able to work on all parts of the project. Making sure there are no regressions ensured that our software was constantly improving and no features were removed. Following a consensus-oriented model ensured all team members had a say in every decision. Distributing the development ensured we had discussions around all code changes, and open communication. By having Short Release Cycles, this allowed for continuous improvement and continuous delivery throughout the software development life cycle.

Implementing these changes was essential to our development. For the future, one practice to place heavier weight on would be the No Regressions rule. For our code base, we had a limited amount of features. For future teams, when they start creating websites, or adding more input formats, it is essential to ensure there are no regressions. Another practice to focus on in the future is Short Release Cycles. When adding more features to our software project, ensuring the release cycles are short will ensure critical functionality reaches end users - or testers more quickly.

REFERENCES

[1] Tim Menzies. 2021. *proj1rubric*. Retrieved September 29, 2021 from https://github.com/txt/se21/blob/master/docs/proj1rubric.md
[2] Jonathan Corbet and Greg Kroah-Hartman. 2017. *Linux Kernel Development Report*. Retrieved September 29, 2021 from https://www.linuxfoundation.org/wp-content/uploads/linux-kernel-report-2017.pdf