Linux Kernel Best Practices and Grading Rubrics

Elizabeth Lin*
North Carolina State University
Raleigh, NC, USA
etlin@ncsu.edu

Neel Shah*
North Carolina State University
Raleigh, NC, USA
npshah6@ncsu.edu

Sanay Shah*
North Carolina State University
Raleigh, NC, USA
sshah34@ncsu.edu

Shaival Shah*
North Carolina State University
Raleigh, NC, USA
sshah35@ncsu.edu

Shivesh Jha*
North Carolina State University
Raleigh, NC, USA
sjha7@ncsu.edu

ABSTRACT

The Linux Foundation had defined the best practices for Linux Kernel development in 2017. This article explores the relation between those best practices and the grading rubrics in the course CSC - 510 (Software Engineering) at the NC State University.

KEYWORDS

Linux Kernel, Best Practices, Software Engineering

ACM Reference Format:
Elizabeth Lin, Neel Shah, Sanay Shah, Shaival Shah, and Shivesh Jha. 2022. Linux Kernel Best Practices and Grading Rubrics. In *NCSU CSC 510, Oct 09, 2022, Raleigh, NC*. NCSU, Raleigh, NC, USA, 2 pages.

1 SHORT RELEASE CYCLES

Related rubric items:

- Short release cycles
- Number of commits

Short release cycles would allow developers to test code changes more frequently. The amount of testing needed for short release cycles would also be smaller. This ensures more and better quality testing for each release. This would also ensure that features can be added periodically allowing both the users and the developers avoid the problems in integrating a large amount features at once in their code.

Commits in regular duration would ensure that small amount of code is pushed every time in the code base, thereby avoiding integration problems that can arise if a large amount of code is pushed.

As release cycles are shorter, there would also be less changes made to each cycle. This would allow easier rollback to a previous release if a change was not satisfactory.

*All authors contributed equally to this research.

2 DISTRIBUTED DEVELOPMENT MODEL

Process scalability requires a distributed hierarchical development model. Related rubric items:

- Workload is spread over the whole team
- Number of commits by different people
- Use of version control tools
- Automated analysis tools
- CONTRIBUTING.MD

As the size of software increases, more and more developers are needed to produce software. This rubric directly relates to the Distributed Development Model practice.

If more people are working on the project, it can be ensured that the Distributed Development Model is followed.

Managing software across a big team can become a challenge. Version control tools ensure that the code can be maintained properly

Automated analysis tools would ensure that each new commit follows some common standard and allows analysis of the code for further optimization.

Managing software across a big team can become a challenge. Clear instructions are needed(like in CONTRIBUTING.MD) to inform each developer what their part is.

3 TOOLS MATTER

Related rubric items:

- Evidence that team is using same tools
- Version control tools
- Style checkers
- Code formatters
- Syntax checkers
- Automated analysis tools

Imagine a large team of developers working on the same project but without version control tools, it would be a nightmare. Version control tools, such as git, have indeed made collaborating on software much easier.

Now imagine using version control, but no checks are being executed when there is a new commit. How do we know if a commit is contributing to the project or if it breaks the code? This is where tools (like style checkers, code formatters, syntax checkers) come in. GitHub actions allows checks and tests to be made before new changes are committed, acting as a safeguard against bad code.

4 CONSENSUS-ORIENTED MODEL

Related rubric items:

- Issue reports
- Issues being discussed
- Issues being closed
- Automated analysis tools
- CONTRIBUTING.MD
- Automated analysis tools
- Version control tools

As more and more developers are involved in a project, more opinions will arise. Discussions among different opinions(Issue Reports, discussion in issues) help the project, and also the developers, grow. Issues on github provide a space for this discussion.

Documentation such as CONTRIBUTING.MD lets developers know how they can make changes to the current software. But not all changes would be approved by the maintainers.

Developers can use pull requests, which allows anyone to contribute, but the maintainers have the final decision on whether or not to approve of the change.

5 THE NO REGRESSIONS RULE

Related rubric items:

- Version control tools
- Style checkers
- Code formatters
- Syntax checkers
- Automated analysis tools
- Test cases
- Issues

To make sure that the code will work as expected on different systems, tools such as GitHub Actions could be used. GitHub actions allow checks for you to make sure that your software will work on different platforms. These tools can check the code for style, format and test cases to ensure that each new commit is of a standard quality.

More discussions in issues and between different contributors can also ensure that there are no regressions in the project.

6 CORPORATE PARTICIPATION

Related rubric items:

- CONTRIBUTING.MD
- Number of commits by different people
- Automated analysis tools

Encouraging contributions from different developers would invite corporate participation in the project. A clear documentation would also help in achieving this. Additionally, automated analysis tools can help the corporate side of the project track and analyze the progress of the project.

7 ZERO INTERNAL BOUNDARIES

Related rubric items:

- Issue Reports
- No. of commits by different people
- CONTRIBUTING.MD

Open source software allows everyone to make contributions. If anyone wants to suggest new changes, they could open an issue on github and discuss it. They could also open a pull request to add new features into the software.

If the maintainers are not taking the discussions or new feature requests into consideration, this would be reflected in discussions. The pressure from discussions would push maintainers to improve their software.

A file like CONTRIBUTING.MD, elaborating how anyone can and should contribute to the project ensures that anyone can jump right into the project if they have any ideas.