Connection Between Rubric Items and Linux Kernel Best Practices (Group 33

Ryan Mertz[*]
rpmertz@ncsu.edu
North Carolina State University
Raleigh, North Carolina, USA

Udith Kalburgi
ukalbur@ncsu.edu
North Carolina State University
Raleigh, North Carolina, USA

Aneesh Sreedhara
asreedh@ncsu.edu
North Carolina State University
Raleigh, North Carolina, USA

Sutapa Dey Tithi
stithi@ncsu.edu
North Carolina State University
Raleigh, North Carolina, USA

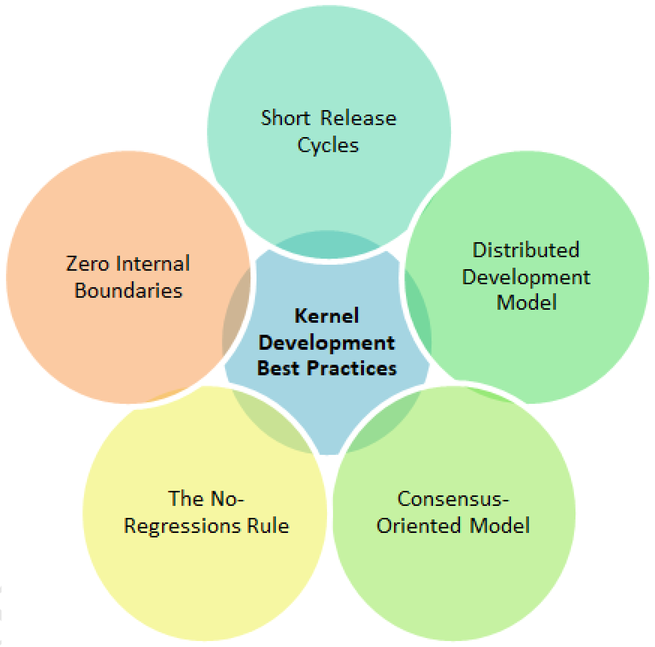

Figure 1: Linux Kernel best practices.

ABSTRACT

This article will review the Linux Kernel best practices and link them to the rubric for project 1 of CSC 510. There will be connections made between the Linux Kernel best practices and the rubric to discover how the rubric follows the guidelines of and evaluates the project in terms of the Linux Kernel best practices.

[*]All authors contributed to this project

CCS CONCEPTS

• **Linux Kernel best practices systems**; • **CSC 510 Project 1 Rubric**;

KEYWORDS

short release cycles, distributed development model, consensus-oriented model, the no-regressions rule, zero internal boundaries

ACM Reference Format:
Ryan Mertz, Udith Kalburgi, Aneesh Sreedhara, and Sutapa Dey Tithi. 2022. Connection Between Rubric Items and Linux Kernel Best Practices (Group 33. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX)*. ACM, New York, NY, USA, 2 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

The project 1 rubric is assessing CSC 510 projects by using a fleshed out rubric that generally follows the Linux Kernel best practices. There are certain items in the rubric that will show evidence for following the Linux Kernel best practices. This article will dive into these specific scoring metrics and unpack how the rubric will be a valid evaluator of the Linux Kernel best practices.

2 LINUX KERNEL BEST PRACTICES

The Linux Kernel best practices include development practices that help guide a system similar to how the Linux kernel was made. These practices are as follows: short release cycles, distributed development model, consensus-oriented model, the no-regressions rule, and zero internal boundaries.

Short release cycles refer to the fast releases of functionality to customers. With short release cycles, new functionality will be keeping users' interests and giving them something to be excited about often. This also takes pressure off of developers as they may not reach deadlines packed with many features, but could focus on one feature for the next release.

Distributed development models split up the development towards people with higher skill sets in certain fields. This model avoids giving one person the whole system to check because they may not be an expert with every part of the system.

Consensus-oriented models make sure that there is no developer that disagrees with the changes that are made so that everyone can be responsible and aware of the system through all of its versions.

The no-regressions rule requires that developers do not backtrack on development and that every stable feature that exists should stay stable and in subsequent releases.

Zero internal boundaries encourages developers to fix problems as they are seen and the changes are justifiable. This improves kernel stability because problems are fixed where they start rather than having to work around another way for this problem to be solved to achieve stability in the system again.

For further information on Linux Kernel best practices, the full definitions are available from https://medium.com/@Packt_Pub/linux-kernel-development-best-practices-11c1474704d6.

3 CONNECTION

The rubric for project 1 is designed to check that the Linux Kernel best development practices are bring used. These practices will determine if project 1 has been developed correctly.

Short release cycles are measured very clearly in the rubric. One of the rubric elements state "short release cycles" and determine if features are being released often enough. The use of version control tools can also be used to evaluate short release cycles because it will be a good quantifier of how often features and new releases are being sent out.

Distributed development models are being evaluated through the rubric's request of workload being spread over the whole team so that each team member is contributing a sufficient amount to the code base. The rubric is also checking for distributed development models by checking that each team member is using the same tools and are all able to run the system and have updated configuration files.

Consensus-oriented models are evaluated in regards to issues being discussed before they are closed and the presence of a chat channel. According to the rubric, issues need to be discussed before being closed, which encourages discussion among the developers to make sure everyone is fine with the changes being made. The presence of a chat channel showcases the developers communicating and making sure that everyone is on the same page. As long as everyone is on the same page, there should be no features that get pushed into the system without every one of the developers knowing about it.

The no-regressions rule can be evaluated through the videos that show new functionality that has been implemented from a previous year's project. The video needs to show that the team took features that were included and added onto them rather than replacing them.

Zero internal boundaries are measured quite a few times throughout the rubric. The rubric evaluates the number of commits by different people so that there is proof that everyone is working on the project and can show that different problems with different areas of the system are being solved no matter who is available to solve it. Documentation is a big part of the rubric and represents zero internal boundaries because of the emphasis that documentation places on letting other developers know what is going on in the code base so that they are more able to contribute. The rubric asks for evidence that developers are working on multiple parts of the code base which is direct connection with the principle of zero internal boundaries. Lastly, the rubric measures that there are plenty of issues being made for failing test cases that can be seen by any developer observing the automatic tests being done on the code.

4 CONCLUSION

The rubric for project 1 of CSC 510 is designed to assess the project's development practices. Using the Linux kernel as a baseline for this assessment is creditable because the Linux kernel was so revolutionary to the computing world.

Received 22 August 2022; revised 4 October 2022; accepted 6 October 2022