Identifying Connections Between Project Rubric and Linux Kernel Best Practices

Seoyeong Park
North Carolina State University
United States
spark43@ncsu.edu

Chris Kastritis
North Carolina State University
United States
crkastri@ncsu.edu

Safaa Mohamed
North Carolina State University
United States
smohame6@ncsu.edu

Collin Riggs
North Carolina State University
United States
cmriggs@ncsu.edu

Soham Bapat
North Carolina State University
United States
sbapat2@ncsu.edu

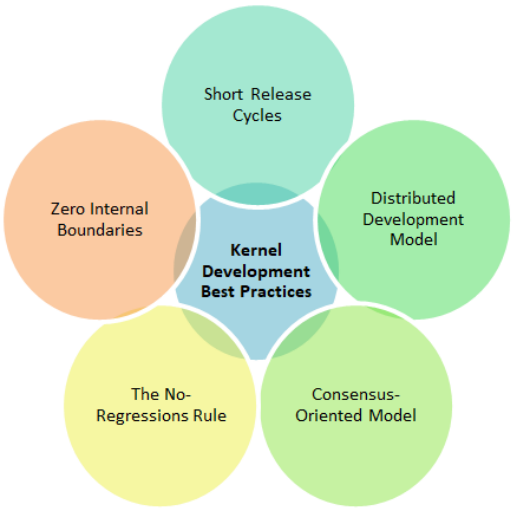

Figure 1: Linux Kernel Best Development Practices

ABSTRACT

This paper goes over the many rubric items for Project 1 in CSC-510 Software Engineering with Dr. Timothy Menzies and shows how each of them relates to the Linux Kernel Development Best Practices [1]. Linux Kernel Development Best Practices are Short Release Cycles, Distributed Development Model, Consensus-Oriented Model, the No Regressions Rule, and Zero Internal Boundaries. In the following sections, we will explain each practice and categorize given rubric to relate it with each practice.

Unpublished working draft. Not for distribution.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
Identifying Connections Between Project Rubric and Linux Kernel Best Practices, October 6, 2022.
© 2022 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/XXXXXXX.XXXXXXX

CCS CONCEPTS

• **Computer systems organization** → **Software Engineering**.

KEYWORDS

software engineering, testing, linux kernel best practices

ACM Reference Format:
Seoyeong Park, Chris Kastritis, Safaa Mohamed, Collin Riggs, and Soham Bapat. 2022. Identifying Connections Between Project Rubric and Linux Kernel Best Practices. In *Proceedings of Identifying Connections Between Project Rubric and Linux Kernel Best Practices.* ACM, New York, NY, USA, 2 pages. https://doi.org/XXXXXXX.XXXXXXX

1 SHORT RELEASE CYCLES

Short release cycles are important as not only it allows for users to get new features without delays but also it maintains stable releases instead of integrating large amount of codes at once. Small projects may not experience short release cycles. In the rubric of project 1, commitment of team members would measure if they contributed enough to their assignments by observing release cycles.

	Project 1 Rubric
1	Short release cycles(project members are committing often enough so that everyone can get your work)

Table 1: Rubrics Related to Short Release Cycles

2 DISTRIBUTED DEVELOPMENT MODEL

Using GitHub allows multiple people maintain the software together. Opening up issues, creating and merging branches, well-written documents help developers maintain quality of their products without crashing it.

	Project 1 Rubric
1	Workload is spread over the whole team (one team member is often Xtimes more productive than the others...
2	Use a version control tool
3	Issues reports: there are many
4	Docs: doco generated, format not ugly
5	Docs: what: point descriptions of each class/function (in isolation)
6	Docs: how: for common use cases X,Y,Z mini-tutorials showing worked examples on how to do X,Y,Z doc page entries
7	Docs: why: docs tell a story, motivate the whole thing, deliver a punchline that makes you want to rush out and use the thing
8	Docs: short video, animated, hosted on your repo. That convinces people why they want to work on your code.
9	Use of code coverage
10	Other automated analysis tools
11	The files CONTRIBUTING.md lists coding standards and lots of tips on how to extend the system without screwing things up

Table 2: Rubrics Related to Distributed Development Model

3 CONSENSUS-ORIENTED MODEL

Consensus-oriented model is close to reporting issues. Once issues are found, they should be shared among developers so that all contributors notice about them rather than a discoverer directly fixes the issues. The reported issues should be fixed and closed after having discussion between developers. As listed in Table 2, rubrics like 'Issues reports: there are many', 'Issues are being closed', 'Issues are discussed before they are closed' are related to this practices.

	Project 1 Rubric
1	Issues reports: there are many
2	Issues are being closed
3	Issues are discussed before they are closed

Table 3: Rubrics Related to Consensus-Oriented Model

4 THE NO REGRESSIONS RULE

The no regressions rule gives users assurance that upgrades will not break their systems. It aligns with controlling versions and testing software for ensuring all features are working fine while updating software.

	Project 1 Rubric
1	Use of version control tools
2	Test cases exist(dozens of tests and those test cases are more than 30% of the code base)
3	Test cases are routinely executed(E.g. travis-com.com or github actions or something)
4	Test cases: a large proportion of the issues related to handling failing cases.(If a test case fails, open an issue and fix it)

Table 4: Rubrics Related to the No Regressions Rule

5 ZERO INTERNAL BOUNDARIES

Zero internal boundaries allows everyone to access the same tools. Since we are working in a public project and every contributor needs to change multiple parts of the code, it is important to get the same tools while developing. Everyone who is involved in developing should be able to install all necessary dependencies with the same version, the same configuration files, and other essential files. In the rubrics, we are evaluated if we use the same tools and files, and if everyone in the team is able to get the same tools and files. Table 3 lists all related rubrics from Project 1 evaluation.

	Project 1 Rubric
1	Evidence that the whole team is using the same tools: everyone can get to all tools and files
2	Evidence that the whole team is using the same tools (e.g. config files in the repo, updated by lots of different people)
3	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)
4	Evidence that the members of the team are working across multiple places in the code base
5	Workload is spread over the whole team (one team member is often Xtimes more productive than the others...

Table 5: Rubrics Related to Zero Internal Boundaries

REFERENCES

[1] Jonathan Corbet and Greg Kroah-Hartman. 2017. *2017 Linux Kernel Development Report.*

draft 6 October 2022

Unpublished working draft. Not for distribution.