Connections Between the Linux Kernel Best Practices and the CSC510 Project 1 Rubric

Heidi Reichert
hreiche@ncsu.edu
North Carolina State University
Raleigh, North Carolina, USA

M M Abid Naziri
mnaziri@ncsu.edu
North Carolina State University
Raleigh, North Carolina, USA

Nikhil Mehra
nmehra2@ncsu.edu
North Carolina State University
Raleigh, North Carolina, USA

Bella Samuelsson
insamuel@ncsu.edu
North Carolina State University
Raleigh, North Carolina, USA

Parth Katlana
pkatlan@ncsu.edu
North Carolina State University
Raleigh, North Carolina, USA

ABSTRACT

Since its release in 1991, the Linux Kernel Best Practices have provided a clear set of useful development practices for the software engineering community at large [1]. A key component of the first project of North Carolina State University's CSC510 first project is a thorough understanding of the use and applicability of these best practices. To that end, this document serves to connect project 1's grading rubric to these practices. Along with a brief introduction to these best practices, each section represents a best practice and a discussion on how that practice connects to the project's rubric item(s) connect.

CCS CONCEPTS

• **Computer systems organization** → **Embedded systems**; *Redundancy*; Robotics; • **Networks** → Network reliability.

KEYWORDS

CSC510, Linux Kernel Best Practices, Project 1

ACM Reference Format:
Heidi Reichert, M M Abid Naziri, Nikhil Mehra, Bella Samuelsson, and Parth Katlana. 2022. Connections Between the Linux Kernel Best Practices and the CSC510 Project 1 Rubric. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 SHORT RELEASE CYCLES ARE IMPORTANT

The use of short release cycles is critical; long cycles mean large amounts of code need to be integrated at once, consequently lending pressure to releasing not-yet-ready code. New, stable code and constant integration help to minimize disruption.

In the project 1 rubric, the "Short release cycles" item explicitly addresses this best practice; however, other items like "number of commits" and "number of commits by different people" also connect to this concept of frequent updates and code accessibility. Frequently updated, functional code is the name of the game for this practice.

2 PROCESS SCALABILITY REQUIRES A DISTRIBUTED, HIERARCHICAL DEVELOPMENT MODEL

Process scalability scales put the responsibility for code review and integration across multiple maintainers. This allows for more coding changes while maintaining quality. One person is not responsible for maintaining the quality of all code.

In the project 1 rubric, "Workload is spread over the whole team (one team member is often Xtimes more productive than the others...," "but nevertheless, here is a track record that everyone is contributing a lot)," "Number of commits," "Number of commits by different people," "Test cases exist," and "Test cases are routinely executed" all connect to this point. The overall idea of workload being distributed among the team connects to the idea of having multiple maintainers; all individuals are working toward this project, and all individuals are checking each others' work (e.g., approving pull requests). The implementation and use of test cases allows for a more streamlined checking of others' work; this distributed development model allows for significantly easier integration of new code into the existing code base, thus reducing the workload of maintaining code quality and, again, preventing only one individual from having to check all code correctness.

3 TOOLS MATTER

The tools that you use matter for development at scale. A smaller-scale project may be able, for instance, to get away without using version control tools; however, any larger-scale project will absolutely not work with this development model. Mistakes happen, things go wrong, and sometimes not all developers in a project can run or access the project in the same way that others do. We need external tools that allow us to keep track of changes, set up projects correctly, and continuously ensure uniformity and correctness of code.

In the project 1 rubric, all the three of the sections relating to "Evidence that the whole team is using the same tools", as well as "Use of version control tools," "Use of style checkers," "Use of code formatters," "Use of syntax checkers," "Use of code coverage,' and

Unpublished work. Not for distribution.

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

"Use of other automated tools" all relate to this idea. The use of tools, from version control to style checkers, all allow for optimal functionality and ease of scalability in development. If all team members are using the same tools, and a variety of tools are being used to ensure correctness of code, then development should be significantly improved.

4 THE KERNEL'S STRONGLY CONSENSUS-ORIENTED MODEL IS IMPORTANT

A proposed change should not be merged if a developer is opposed to it. This ensure that the software remains suited to a wide range of users/problems, and no one user can make changes at the expense of others. This allows for a greater range of functionality, and prevents potential development dictators from controlling the code.

In the project 1 rubric, "Chat channel: exists" and "Ideas are discussed before they are closed" relate to this practice. These rubric items allow for democracy in coding; specifically, they ensure communication and an allowance for disagreement. Pull requests, for instance, can be discussed or rejected (obviously within reason).

5 A RELATED FACTOR IS THE KERNEL'S STRONG "NO REGRESSIONS" RULE

Upgrades, in general, should not break your existing code; if one thing works in a specific setting, subsequent developments must also work there. By using this principle, the users of the code can ensure that, even as new features and critical upgrades are implemented, their existing functionality will remain.

In the project 1 rubric, "The files CONTRIBUTING.md lists coding standards and lots of tips on how to extend the system without screwing things up," as well as all test-related items, address this practice. A specific file demonstrating coding standards and options for extensions allows for developers to both understand the existing standards in place (and thus not violate existing code), as well as understand what options exist for them in terms of extending the code (as opposed to limiting it). The existence of tests, as mentioned before, also allows for the effective testing of compatibility of new code with the existing project.

6 CORPORATE PARTICIPATION IN THE PROCESS IS CRUCIAL

In an industry setting, the participation of your clients is crucial in software development; this provides not only capital/funding, but also motivation. Another critical component is that one corporation's suggestions for development can harm or restrict what the software can do; in other words, while anyone can recommend improvements for their own needs, nobody can take away functionalities for others.

Because project 1 was not developed in conjunction with any company's needs or in answer to any specific prompts, we do not believe that any specific rubric item addresses this practice.

7 THERE SHOULD BE NO INTERNAL BOUNDARIES WITHIN THE PROJECT

Developers have their areas of expertise and focus; however, the concept of having no internal boundaries allows for all developers to make any changes to any parts of the code, provided there is justification for such changes. Because of this, problems can be fixed where they originate, rather than being worked around. Also, because all individuals can access all areas of the code, a more holistic view of the code can be taken, allowing for maintainers to understand most (if not all) parts of the code.

In project 1, rubric items "Evidence that the members of the team are working across multiple places in the code base," "Number of commits" and "Number of commits by different people," as well as all of the "Everyone uses the same tools" rubric items address this practice" and "Workload spread" address this practice. Effectively, all members of the team ought to be able to point to different areas in the code base and show that they've been contributing (or, at least, have the ability to contribute); this indicates a lack of specified project boundaries per developer. Likewise, the rubric items "Issue reports: there are many" and "Issues are being closed" point to equal opportunity for developers to be able to see both what is going wrong around the code and also understand what they may contribute to in different areas of the project. Three docs points regarding "Point descriptions of each class/function," "Automatic documentation," and "Common use cases X,Y,Z mini-tutorials showing worked examples on how to do X,Y,Z" also suggest giving not only users, but also developers, an understanding of how every component in the code is working (and, thus, where they may contribute).

8 OTHER RUBRIC ITEMS AND THEIR IMPORTANCE

A few rubric items do not rear their heads in the aforementioned sections. It is worth noting their importance in the realm of development here.

The presence of a DOI badge, one of the rubric requirements, is significant, both for the purposes of record and documentation; a unique document identifier allows for a project to be traced over time and strongly connected to its developers. The use of a punchline/motivation in part may relate to corporate participation, but more strongly connects to the idea that the projects we work on should matter; they should have meaning and significance behind them. A short video demonstration of how it works may be part of that (although it also provides use in helping users understand what the final code result should look like).

Overall, projects should be well-documented, easily understandable, and updated in short bursts so as to maximize their utility and optimize their functionality.

REFERENCES

[1] Jonathan Corbet and Greg Kroah-Hartman. 2017. 2017 linux kernel development report. *A Publication of The Linux Foundation* (2017).

Received 7 October 2022