Connection Between Rubric and Linux Kernel Best Practices

Guanyu Hou
NC State University
Raleigh, U.S.
sixgod666@gmail.com

Rundi Liu
NC State University
Raleigh, U.S.
rliu26@ncsu.edu

Haoqu Ma
NC State University
Raleigh, U.S.
mahaoqu@gmail.com

Ioshua Lin
NC State University
Raleigh, U.S.
linjos6750@gmail.com

Zhijin Yang
NC State University
Raleigh, U.S.
zhijinyang@outlook.com

ABSTRACT

The paper is to illustrates the connection between our project1 and Linux kernel best practices. Our team was trying to apply those practices and we will explain how our team connect those practices through this paper.

ACM Reference Format:
Guanyu Hou, Rundi Liu, Haoqu Ma, Ioshua Lin, and Zhijin Yang. 2021. Connection Between Rubric and Linux Kernel Best Practices. 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

Short release would be extremely important for some very huge projects. If they do not follow short release cycles, releasing new version would be very stressful, because huge amounts of code had to be integrated at once. In our team, we always commit and push in time and frequently, which can make our project more stable. It reduces the probability that the project cannot run and team members can handle the new code easily. At the same time, our efficiency would be improved!

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

2 PROCESS SCALABILITY REQUIRES A DISTRIBUTED, HIERARCHICAL DEVELOPMENT MODEL

The quality of "task-related processes" appeared to determine whether distributed teams became a "liability or an opportunity." We definitely followed this, because there is a rule on rubric of project 1, which says each member have to contribute the project in same level, it is not allowed that some team members commit and push a lot, and someone contribute a few to the project. We respected this in these ways; firstly, everyone has his or her own job, and those work would be fair logically, because we need to ensure each team member contributes our project in same level. Secondly, spreading out the responsibility can increase our efficiency. The power of one people is insignificant, we would be five times more efficient if we had the same ability, since we have 5 team members. Lastly, the advantage is that everyone feels involved, and everyone can understand how the project works. That's very important, because we may confront any situation. For example, if the project crashes or goes wrong and we want to fix that problem in time, and there is only one developer is available. If he does not know how the project works, then he does not know how to fix the problem.

We contributed our project based on this, as so far, we have 74 commits, no one contributes much, and no one contributes little. Since everyone contributed equally logically, all team member know the methods and classes. One of team members cannot solve bugs, other team member always can help and solve them together.

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

In our project1, we did not regress, because the time is limited. It is hard to develop a full-featured or multi-functional. We are not familiar with Android Developing, so that we learned and developed. Since the project does not have many functions, there won't be huge amounts of code, and we only have 3 weeks. So, we haven't have a chance to regress. And we will also avoid regression in the future.

4 THERE SHOULD BE NO INTERNAL BOUNDARIES WITHIN THE PROJECT

Zero-internal boundaries is defined by developers fixing problems they come across, even if it is not part of the specific section they work on in the kernel. This can help a developer understand the whole kernel better and allow developers to fix problems rather than write code that works around them. In the rubric for project 1, we can see many ways in which the items in the rubric reflect zero-internal boundaries. The first is issue reports; issue reports can allow developers to see issues that other developers are having trouble with or even parts of the code that needs further development. This can allow a developer to work on other parts of the kernel which can in turn help them understand the whole kernel. In addition, a developer can fix a known bug that affects a section of the kernel before working on code related to that same section of the kernel. In addition, part of help zero-internal boundaries is discussing issues before closing them and having chat channels. These both help facilitate discussion about the code and issues, which can help developers have a wider view of the kernel as a whole and understand it better. Discussing issues before closing will especially help developers understand the fixed bug and may help in pointing out new bugs that were not seen before.

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

The kernel's strongly consensus-oriented model is important. As a general rule, a proposed change will not be merged if a respected developer is opposed to it. In the project 1 rubric, issues are discussed before they are closed would strongly support this point. Our teammates can assign their goal after enough discussion. With a few meeting time, many code conflicts will be avoided.

6 TOOLS MATTER

before starting project1, we wanted the project is written in Python, but some of us do not know Python much, and we found everyone is able to handle Java. And everyone is using Android Studio as IDE and emulator on IDE. It would be easy to communicate some problems we confront in terms of setting up an environment.

Tools are matter, because tools can help us develop the project. For project1, we are using a good IDE, we do not need to manually enter every file and check for synatax. And a good IDE can visually help us work faster, due to its dark mode, which can make us focus on the structures.