Project 1 Rubric Comments

Janet Brock
North Carolina State University
Raleigh, NC, USA
brockjd@ncsu.edu

Shahnewaz Leon
North Carolina State University
Raleigh, NC, USA
sleon3@ncsu.edu

Dong Li
North Carolina State University
Raleigh, NC, USA
dli35@ncsu.edu

Drew Cummings
North Carolina State University
Raleigh, NC, USA
docummin@ncsu.edu

Cheng-Yun Kuo
North Carolina State University
Raleigh, NC, USA
ckuo3@ncsu.edu

ACM Reference Format:
Janet Brock, Shahnewaz Leon, Dong Li, Drew Cummings, and Cheng-Yun Kuo. 2022. Project 1 Rubric Comments. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 2 pages. https://doi.org/10.5281/zenodo.7155414

1 INTRODUCTION

The Linux Kernel is one of the most amazing pieces of open-source software. Originally created in 1991 it has continued to be used and relevant for 31 years. To understand how the Linux Kernel has been so successful you can look at the Linux Kernel Best Practices; a list of practices that has kept the Linux Kernel to a high standard for all these years. The rubric for our Project 1 can also be read as a list of requirements aiming to keep us to a high standard. When comparing the rubric to the Linux Kernel Best Practices you can find many connections that help ensure our project lives up to the amazing standards set by the Linux Kernel. By tracking these connections we can see how the rubric teaches us important software engineering lessons and encourages us to be better programmers.

2 LINUX KERNEL BEST PRACTICES CONNECTIONS

One of the key practices of the Linux Kernel Best Practices is Short Release Cycles. Short release cycles are important because they mean new features get to users sooner, huge amounts of code don't have to be integrated all at once, and developers don't feel pressure to merge code before it's ready since there will always be another release in the near future. In the Project 1 rubric there is specifically a category for short release cycles, which emphasizes just how important short release cycles are to making good, usable software. There is also a section on number of commits, which is another more quantifiable measure of whether or not we are living up to the standard of short release cycles because the more commits the more likely those commits were soon after another, meaning the more likely the release cycles were short.

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

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.5281/zenodo.7155414

The second key practice of the Linux Kernel Best Practices that we can see reflected in the Project 1 rubric is Distributed Development Model. A distributed development model is important because it means one person doesn't get stuck with an unreasonable amount of work; by distributing the work between many people you can keep up with demands as your software grows. We can see this idea reflected in the rubric through the requirement that the workload is spread over the whole team and that each team member has a large number of commits. Other categories that connect to the concept of a distributed development model include having clear standards in CONTRIBUTING.md because this allows many people to work on the software which further distributes the work. The requirement to have evidence that the whole team is using the same tools means everyone can easily contribute to the project, and the more people able to contribute the more you can distribute work. Lastly, evidence that team members are working across the code base shows that work is being distributed because team members are doing lots of different work within the project.

The next important practice of the Linux Kernel Best Practices that we can see reflected in the rubric is Consensus-Oriented Model. Having a consensus-oriented model is important because no particular group or person can make changes to the software at the expense of anyone else; it keeps the software fair and usable by anyone. We can see the consensus-oriented model encouraged in the rubric with the requirement that all issues are discussed before they are closed because the discussion around closing an issue will involve all team members agreeing the issue is ready to be closed, thus they must reach a consensus. The requirement that a chat channel exist also encourages communication which is key to reaching consensus. The consensus-oriented model can also be tied to the requirement that all team members use the same tools because a consensus has to be reached about which tools to use.

The rubric also has clear connections to the Linux Kernel Best Practice of Zero Internal Boundaries. Having zero internal boundaries is important because it means problems to be fixed at their origin rather than worked around, developers will have a broader understanding of the software as a whole, and no one person can stall progress for everyone else. The rubric clearly embodies this ideal with the requirement that team members work multiple places across the code base. Zero internal boundaries is also encouraged by the requirement that CONTRIBUTING.md be clear about how people can contribute because this will provide anyone with clear directions about how to contribute anywhere in the code base. Additionally the requirement that all team members use the same tools

makes it easier for team members to contribute anywhere in the code base. To an extent the requirements that workload is spread across the team and that all team members have many commits also encourage zero internal boundaries by encouraging team members to take on lots of different work across the repository.

The last standard Linux Kernel Best Practice that the rubric encourages is the No-Regressions Rule. This is an important rule because it gives users assurance that new features and updates will not break the software for them because it requires that if the software works in a specific setting, all subsequent versions of the software must work there too. Though this may not be as clear of a connection, the rubric still encourages this practice with the requirements that there are many test cases that are routinely executed. Those requirements ensure that the software is always working as it should be and will not need to be regressed.

Though not always included in the Linux Kernel Best Practices there is another practice sometimes added that Tools Matter. This is important because without good tools it would be almost impossible to create good, complex software. This is clearly emphasized by the rubric with the requirement that version control tools, style checkers, code formatters, and syntax checkers are used and that all team members use the same tools. By requiring all these different tools the rubric ensures that it will be possible for the teams to build quality open-source software and emphasizes the importance of having and using good tools.

3 CONCLUSION

It is clear the Project 1 rubric is tailored to encourage following the Linux Kernel Best Practices. A high number of commits lead to short release cycles, having workload spread across the team leads to a distributed development model, issues discussions and a chat channel inspire inspire use of a consensus-oriented model, team members working across the code base leads to zero internal boundaries, test cases make following the no-regression rule easier, and requiring many different tools emphasize that tools really do matter. To truly follow this rubric would necessitate using the Linux Kernel Best Practices. By following this rubric the value of the Linux Kernel Best Practices would become clear to anyone; which will hopefully inspire us and other students to use it going forward in our lives. The Linux Kernel was made 31 years ago, maybe 31 years from now one of us will have our own piece of long lasting open-source software that future rubrics and students can be inspired by. Together we can follow the Linux Kernel Best Practices and pave the way for a brilliant future of open-source software.

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009