How Right is Write Right

Saikaushik Kalyanaraman
skalyan2@ncsu.edu
North Carolina State University

Shakthi Nandana Govindan
North Carolina State University
sgovind4@ncsu.edu

Aswin Itha
North Carolina State University
aitha@ncsu.edu

Ashwin Shankar Umasankar
North Carolina State University
aumasan@ncsu.edu

Kailash Singaravelu
North Carolina State University
ksingar@ncsu.edu

ABSTRACT

A clear and well-documented LaTeX document is presented as an article formatted for publication by ACM in a conference proceedings or journal publication. Based on the "acmart" document class, this article presents and explains many of the common variations, as well as many of the formatting elements an author may use in the preparation of the documentation of their work.

CCS CONCEPTS

• **Software Engineering**; • **Linux Kernel Best Practices**;

ACM Reference Format:
Saikaushik Kalyanaraman, Shakthi Nandana Govindan, Aswin Itha, Ashwin Shankar Umasankar, and Kailash Singaravelu. 2021. How Right is Write Right. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The Linux Kernel is a prime example of open source and the merits of constructive collaboration. Over the period of 27 years, there have been 14000+ contributing developers all over the globe. Despite the huge number of developers(and the ensuing chaos), the linux kernel's longevity and success are a thing to behold. Therefore we look at some of the best practices which are :

1)Short Release Cycles
2)Distributed Development Model
3)Consensus-Oriented Model
4)The No-Regressions Rule
5)Zero Internal Boundaries

Now we look at how we, the developers of Write Right, adhered to these principles while developing our project.

2 LINUX KERNEL BEST PRACTICES

Next, we discuss each of the above mentioned practices in detail and how we applied these practices while developing our software.

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

2.1 Short Release Cycles

In the initial days of Kernel development, major releases were made once every few years. This means that major chunks of code would be replaced every time which is inefficient and caused huge issues. So, the development community switched to Short release cycles, which addressed all the issues with the long release cycle. New code is immediately integrated resulting in a stable release. This allowed the release of fundamental changes without causing major issues.

Some metrics in the rubric associated with short release cycles:

1. Use of version control tools: We used Git, a version control tool, to commit each developers work immediately.

2. Using branches and commits: Each developer had their branches with their work and we tracked and merged it using commits and Pull requests.

As this is early stages of the project, we haven't had any releases yet.

2.2 Distributed Development Model

The Linux kernel's code base, with it being and Operating System Kernel, was complex and varied. Hence it was impossible for one person to keep track of the entire code base. The Linux development community quickly realized this and moved on to a distributed model. This meant assigning different portions of kernel to different people based on their expertise. This meant, the code review and integration was seamless, no matter the number of different areas in the kernel.

Some metrics in the rubric associated with Distributed Development Cycles :

1. Chat Channel: exists : We had a dedicated channel to discuss the software requirements and implementation challenges.

2. Issues are discussed before they are closed : We realize the importance of the issue tab in github, but we had most of our meetings in person to discuss outstanding issues in person, therefore eliminating the need of the issues tab. Once an iteration of our product was up and running, we had bug bash sessions to test out the software and find the issues at that point in time. Once we were done with that, we assigned it to respective team members and solved it together.

3. Workload is distributed across the entire Team : We split the work evenly across the group and we classified it into a few areas of discipline. If two team members were assigned a particular class of code, one member would be directly responsible for reviewing the other member's group.

2.3 Consensus Oriented Model

The Linux kernel community strictly adheres to the consensus-oriented model. This states that no single group can make changes to the code base at the expense of the other groups. This makes sure that any and all proposed changes are discussed by the developers and are implemented after a consensus. This is a large part of a good development model, as this ensures all developers share both responsibility and contribution. some metrics in the rubric associated with Consensus oriented model

1. Source code is stored in a repository under revision control: Our project is available in Github and all developers are added as contributors.

2. Issues are discussed before closing: We met twice every week to discuss the progress and address any issues. As we raised any and all issues in person, we dint use the issues feature in Git, but we recommend doing so if you can in the future. We discuss the raised issues in the meetings. We agree on the resolution and direction of development and then share the load among us based on our areas of strengths.

3. Support emails sent to multiple people: We have created a separate email id for the project and the whole team as access to it. So any email regarding the project sent to the email id will reach all the contributors.

By following Consensus oriented model, all developers were able to contribute to the project and have a say in the direction of development.

2.4 No Regressions Rule

The linux kernel code base constantly evolves, and certain versions of the kernel work in very specific environments, but the linux community makes sure that every subsequent version of the kernel work in those environments as well. Also if they find out that the system is being affected by regression, then they waste no time in rolling back to the most stable version.

Some metrics in the rubric associated with Distributed Development Cycles :

1.Test Cases Exist

2.Test Cases Are Routinely Executed

3.The files CONTRIBUTING.md lists coding standards and lots of tips on how to extend the system without screwing things up

Throughout the course of our development process, we always make sure the main branch is bug free and the most stable version of the code. Any new features are extensively tested(manually) across multiple browsers and operating systems before we check in to the main repository. We have expressed our testing process in some detail in our github repository. Therefore for all future releases, we will naturally be equally vigilant and make sure not to release partially tested code, ensuring no regression.

2.5 Zero Internal Boundaries

The Zero internal boundaries practice enables developers to make changes in any part of the code, as long it's justifiable. This is ensured by making sure all developers have access to the tools used in the development of the software. It also enables them to run the software in their local system and work on any part of the application as applicable. Moreover, it also gives the developers a wider view of the kernel as a whole.

some metrics in the rubric associated with Consensus oriented model:

1. Evidence that the entire team is using the same tools: Everyone had access to the entire code base and the setup to contribute to the same.

2. Evidence that the members were able to work on multiple parts of the code base: Everyone can work on any part of the application.

The entire team is added as contributors. All the contributors had the code base in their local machine and worked on different parts of it. As our project is a chrome extension, after every change the extension is updated and is accessible to all the developers to clone and check the functionalities in their machine. The details on how to run this are present in the readme.md. As the contributors of the project, the entire team has access to all the files and folders in the repository.

3 CONCLUSION

As discussed in the document, our project followed the Linux kernel best practices and adhered to the industry's best practices. It helped the development model in multiple ways. By having a consensus-oriented model, all the contributors had a say in the development. By sticking to shirt release cycles, we were able to minimize changes to the core code and avoid huge issues. By following distributed development model ensured we had open discussions about the direction of the development. With Zero internal boundaries, all the contributors were able to access the project and work on any part as applicable. The no regression rule helped in maintaining the software stable with multiple manual testings. The future scope includes automate testing which will make the system run more smoothly.

The roadmap.md file has all the details about our projected future scope. Please adhere to the best practices discussed above when implementing new features.

REFERENCES

[1] Tim Menzies. 2021. *proj1rubric.* Retrieved September 30, 2021 from https://github.com/txt/se21/blob/master/docs/proj1rubric.md

[2] *https : //medium.com/@Packt_Pub/linux − kernel − development − best − practices* − 11c1474704d6