‘Spark’ Development Upholds Linux Best Practices

Natalie Kraft
nakraft@ncsu.edu
North Carolina State University
Raleigh, North Carolina, USA

Siddarth Royapally
sroyapa@ncsu.edu
North Carolina State University
Raleigh, North Carolina, USA

Parth Sarthi
psarthi@ncsu.edu
North Carolina State University
Raleigh, North Carolina, USA

Ashrita Ramaswamy
aramasw@ncsu.edu
North Carolina State University
Raleigh, North Carolina, USA

Jyothi Sumer Goud Maduru
jmaduru@ncsu.edu
North Carolina State University
Raleigh, North Carolina, USA

Kanishk Harde
knharde@ncsu.edu
North Carolina State University
Raleigh, North Carolina, USA

Figure 1: Spark Demo Marketing Animation, 2022.

ABSTRACT

While pursuing the development of a hands free video conferencing software, the authors looked to ensure the Linux Kernel best development practices were upheld. The following documentation establishes that the team recognizes the value of these principles and was intentional about following the ideal practices throughout the development cycle.

CCS CONCEPTS

• **General and reference** → **Computing standards, RFCs and guidelines**; • **Social and professional topics** → Sustainability; *Project management techniques.*

KEYWORDS

linux kernel, development practices, open source, standards

CSC 510 Fall ’22, October 8, 2022, Raleigh, NC

https://doi.org/10.5281/zenodo.7157874

ACM Reference Format:
Natalie Kraft, Siddarth Royapally, Parth Sarthi, Ashrita Ramaswamy, Jyothi Sumer Goud Maduru, and Kanishk Harde. 2022. ‘Spark’ Development Upholds Linux Best Practices. In *Proceedings of Project 1 Rationale for Upholding Linux Best Practices (CSC 510 Fall ’22).* ACM, Raleigh, NC, USA, 2 pages. https://doi.org/10.5281/zenodo.7157874

1 SHORT RELEASE CYCLES

In the early stage of development, Spark had only 1 version released. However, the short release cycle practice was still upheld through the frequent commits occurring through development. This ensured that members working in parallel could create branches to continue development. Frequent commits allowed developers to integrate less code into their branch at once, which improved the number of regressions that occurred due to developing new features. Utilizing Git Insights, it can be seen that the team utilized over 100 commits to establish the first version of Spark.

Moving forward, as there is now a stable version of Spark released, it is expected that versions will be released every 3-6 weeks during which time new features can be added. These short releases will allow developers to not have as much pressure to release a version which is not ready and decreases the risk of regressions due to having version updates that are too large.

2 SCALABILITY

To support continuous development and minimize regressions, our team used a hierarchical development model. Each member was responsible for an area of the development that could occur in parallel - specifically, our areas of responsibility were frontend, user experience (volume and screen sharing), sockets, and gesture recognition. To support code review and integration, we looked to spread out this responsibility between our team members to ensure that review requests could occur frequently. This distribution can be seen within the GitHub Insights, demonstrating that everyone was engaged in areas of the code base that could be developed in parallel.

Furthermore, the responsibility for integration between different branches and functionality was spread out by different developers. This can be seen through looking into the distribution of members who opened pull requests; 6 team members were responsible for opening up over 30 pull requests to integrate these different elements.

3 TOOLS MATTER

Utilizing Git's version control ensured that components could occur in parallel and that integration of these areas would be easy and efficient. Within Git, the project board, issue tracker and code analysis tools were used for further project management. The project board helped to track responsibilities and long term goals; the issue tracker demonstrated identified bugs and feature requests; and the code analysis tools provided consistency across code styles and increased security. Without these tools, Spark would not have been able to maintain the set pace of development.

To ensure that proper practices were being upheld, the team looked into resources regarding the best git practices to utilize [1, 3]. This ensured that prior to development, team members were aware of the principles they should be following when utilizing the project board, issue tracker and development branches.

4 CONSENSUS ORIENTED MODEL

The consensus-oriented model ensures that proposed changes are not merged if another developer is opposed. In Spark, several PRs were rejected with comments left regarding why the change should not be made. Some rejected changes were large, including a disapproval of a particular third party software being utilized to share screens. Other rejected changes were minimal due to improper code style or automated code analysis tools that were not yet passing. Per the consensus-oriented model, rejecting these requested merges ensured that no functionality was harmed from another feature's development. Throughout the development of spark, 4 pull requests were closed, showing that the team upheld this practice. Developers were encouraged to update their code base to solve the disagreement and then could resubmit the request.

5 NO REGRESSIONS

As Spark was still in the early stages of development, with no versions yet released during this phase of the project, we had yet to reach a time where we needed to ensure no regressions occurred within a stable branch of the program. However, this no regressions rule was followed when looking to merge branch functionality. For example, take a look at PR 34 (https://github.com/SiddarthR56/spark/pull/34) which was closed due to failing functionality to not disturb another branch. One developer even commented on this potential regression in the repo stating that "We have some conflicting files that need to be parsed and tests must be passing in development before a move to main should occur." Prior to completing a PR, conflicting files were rerun through blackbox tests to ensure the functionality still was working. This provided developers with assurance that their branch would still function even after the addition of new features propagated by a different developer. Moving forward, the no regression rule will be especially important to ensure that any additional features do not break existing functionality.

6 PARTICIPATION

This project was dependent on several other libraries to support development. While we were able to adapt our code to fit our needs, we are not allowed to change these libraries as it would be harmful to other developers. It is similar to the linux kernel. No changes can be made that would hurt other players. In developing Spark, we aimed to uphold this principle by getting all of our team members involved in development and ensuring that no one branch's feature development would be a detriment to someone else's development. While this is a small scale in comparison to the linux kernel's development, it is a valuable lesson that participation should have broad access and high standards for implementing changes so that the software can serve all.

7 NO INTERNAL BOUNDARIES

Throughout Spark development, everyone had access to the entirety of the project. In addition, config files were actively being updated and shared so that development environments could match. For example, take the package-lock.json file; it establishes versions of all the needed dependencies and was updated 18 times by 5 people throughout the project's development. This type of collaboration shows that everyone had access to the project and could support the development anywhere that the change could be justified.

To further ensure no internal boundaries existed, the developers met frequently to describe their area of the code and act as a SME for any new technology they were introducing. This ensured that people with different skill sets were being brought up quickly on new developments. Specifically, this occurred with the base implementation using sockets and the ML model for gesture recognition.

REFERENCES

[1] Don Batory and Sean O'Malley. 1992. The Design and Implementation of Hierarchical Software Systems With Reusable Components. *ACM Transactions on Software Engr. and Methodology* (Oct. 1992). https://www.cse.msu.edu/~cse870/Materials/Frameworks/tosem-92.pdf

[2] Jonathan Corbet and Greg Kroah-Hartman. 2017. 2017 Linux Kernel Development Report. *The Linux Foundation* (2017), 1–29. https://go.pardot.com/l/6342/2017-10-24/3xr3f2/6342/188781/Publication_LinuxKernelReport_2017.pdf

[3] Sanket. 2019. Best practices for using Git. *Deep Source* (Feb. 2019). https://deepsource.io/blog/git-best-practices/

Received 9 October 2022; revised 9 October 2022; accepted 9 October 2022