Youtube Scrivener

Pragna Bollam
pbollam@ncsu.edu
North Carolina State University
Raleigh, NC, USA

Bhavya Omprakash Agrawal
bagrawa@ncsu.edu
North Carolina State University
Raleigh, NC, USA

Anshul Navinbhai Patel
apatel28@ncsu.edu
North Carolina State University
Raleigh, NC, USA

Rohan Jigarbhai Shah
rshah29@ncsu.edu
North Carolina State University
Raleigh, NC, USA

Darshan Manharbhai Patel
dpatel35@ncsu.edu
North Carolina State University
Raleigh, NC, USA

ABSTRACT

The Five Linux Kernel Development best practices are described along with how these practices are streamlined in the development of the Youtube Scrivener project. The connections between project1 grading rubric [1] and Linux Kernel best practices are detailed.

CCS CONCEPTS

• **Software Development**; • **Linux Kernel Best Practices** → *Software Development Pratices*;

KEYWORDS

Software Development, Linux Kernel, Development Practices, Youtube Scrivener

1 INTRODUCTION

Linux is an open source collaborative project development platform. The lowest level of software running on Linux is the Linux kernel which maintains the hardware, user programs, overall security and integrity of the system. This Linux Kernel is effective and is dominating other open source project development platforms because of the practices employed in its implementation. These best practices are employed in every software project management cycle to increase the efficiency and thereby improving the usage. The Linux Kernel development best practices are:

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

These Linux Kernel best practice are employed in the project Youtube Scrivener following the rubric from project 1 grading rubric [1]. These connections are explained in detailed.

2 LINUX KERNEL BEST PRACTICES

2.1 Short Release Cycles

Short release cycles are essential in a software development project as it helps bring out small features to the user quickly. Short releases means that the developers are constantly working to give the best experience to the user. Also, when there are long release cycles the pressure to integrate huge amounts of code is high which can lead to failure. Therefore, short release cycles guarantees stable release to the software. Short releases ensures short term goals defining the timeline perfectly and ensures the developer that if a release is missed another one is going to be in a few months and not after a very long period.

Short Releases helps the developers to change the flow of the software at any release and also allows them to incorporate new technologies which may not have been available at the original release.

Since the timeline for this project was of a month, releases were done as frequent multiple push requests to the project.

2.2 Distributed Development Model

Each project needs to be developed by many developers to review and develop quality of a project. A distributed model divides the projects into various small tasks and assigns them to different developers. With this model, each developer can work independently and propose changes to the system instead of one developer working on a project. This way the project will be developed with extensive reviews, debugging and improved efficiency. Overall performance of the project can be increased.

The project 1 grading rubric [1] mentions that the workload should be distributed among the group members to be developed. In this project, the work load is equally distributed among all the team developers which can be seen through the number of commits and issues handled by each contributor. The rubric [1] also mentions that group meetings and having a communication channel can help achieve a distributed model. A Slack communication channel was created for the discussion of project updates. Weekly zoom/in-person meetings are conducted to discuss and assign tasks. So, a Distributed Development Model is employed for this project.

2.3 Consensus - Oriented Model

Consensus - Oriented Model of development means that when any change is made to the project code, it is done with the agreement of majority contributors of the project. This makes sure that whenever an update is released no feature or part of the code is compromised and also helps the developers verify the correctness of the code to be added. This model ensures that the software does not fail due to code with error.
During the development of this project the developers of the software scheduled meetings to discuss the open issues before they

were resolved and any changes to the code is done. This discussion ensures that the solution for the issue is approved by the other developers and also that the code does not conflict with any pre-existing code. At any given point there were at least two contributors assigned to check any code developed for any open issue before it is being pushed to the Git repository. Github checks for any conflicts between multiple push and a person is assigned to go over the conflicts to ensure quality control. Many test cases have been applied to issues to make sure that there is no error to the code.

2.4 The No-regression Rule

Any project may have bugs and these bugs must be removed from time to time. So, removing these bugs involves adding extra software code or some internal changes. But adding these changes should not reduce the existing quality of the software.

The No Regression Rule in the Linux Kernel development best practices means that any upgrade to the code or system should not break their system. Any update should not reduce the quality. Any subsequent kernels should work in the same setting as the main kernel.

In this project, corresponding to the guidelines in the project 1 grading rubric [1], all of the version updates or any debug updates are added in the documentation.So, anyone can check the documentation and get to know about the updates. The documentation includes revision control of source control, versions, key features, new releases(if any), etc. From this documentation, it is clear to find out about any update that improves or reduces existing quality. From this, we can see that the this project has employed the No regression Rule and so any user can check details of any updates and its impacts.

2.5 Zero Internal Boundaries

Each project is developed by different developers and each developers should have access to all parts of the project. The tools and code should be accessed by everyone developing the project. But if a developer is unable to access any part of the project to wither review code or develop it, the efficiency of the distributed model reduces. So, any limitations over the access of the resources should be taken care in order for a good performance model.

A zero Internal Boundaries mean that the resources, tools, code and any other aspect of the project should be available to everyone developing the project i.e., no boundaries or limitations.

The project 1 grading rubric [1] connects to this best practice as "evidence that the whole team is using the same tools" and "evidence that the members of the team are working across multiple places in the code". In this project, there is no boundary to access any part of the project anywhere to any developer. Each developer can access the code from any part of the project through the cross-platform open-source tools. All developers of the project have the same accessibility to all the files in the platform.

3 CONCLUSION AND FUTURE WORK

The Linux Kernel best practices were understood and some of those practices were implemented during the development of the project. The practices were not difficult to understand or to implement but a few of them weren't completely followed due to the short duration of the project. Consensus- oriented and Zero internal boundaries were some of the practices best followed for this project. There was complete transparency maintained among the collaborators during the development of the project and everything is publicly available. Clear and correct communication was maintained and inputs and suggestions from all the members was received.

One of the practice which can be worked upon was the short release cycles as the duration of this project was short multiple releases to the project was not done. We aim to achieve this by other contributors who might work on this project to add more features to the project which will add multiple releases to this project.

REFERENCES

[1] Tim Menzies. 2021. Project 1 Grading Rubric. (2021). https://github.com/txt/se21/blob/master/docs/proj1rubric.md