Linux Kernel Best Practices

Ekansh Singhal
Dept. of CSC
esingha@ncsu.edu

Rahul Rangarajan Kannan
Dept. of CSC
rrangar@ncsu.edu

Gowtham Sathyan Arulselvan
Dept. of CSC
garulse@ncsu.edu

Supriya Krishna
Dept. of ECE
sbkrishn@ncsu.edu

***Abstract*—This paper aims to reflect on the various Linux Best Practices and its incorporation in Project 1. Our project, Job Trackr is an application tracking system in which candidates can manage their job applications including job profile, applications status, important dates, notes, saved applications, job descriptions and more. We implemented the Linux best practices as we worked on the project.**

I. Keywords

Linux, best practices, development, application, GitHub

II. Introduction

The Linux Kernel Report published in 2017 provides instances of best practices that contributes to the successful development of a project. These practices are commonly referred to as the "Linux Best Practices." The graduate curriculum for software engineering comprising of Project 1 rubric is on par with these ideals. We will thus provide details on how these best practices were implemented in project "Job Trackr".

III. Comments on Best Practices

The below section comprises of detailed descriptions of Linux Best Practices and how our project aligns with the same. This document will also explain the major examples that showcase these practices implemented.

A. Short Release Cycles

In the early days, major release happened every few years. This meant that the users had to wait for a very long time before a new feature was released. However, now most companies have moved to Agile methodology with short release cycles. This means that the amount of code that should be integrated is relatively small and the new features are readily available to the users.

Since the duration for Project 1 was relatively short, it was extremely important that we had short release cycles. As a result, we were able to quickly find the issues in the project and resolve them. We worked on front end design and backend APIs simultaneously. So it was important that we continuously commit the changes to the repository so we could seamlessly integrate the frond end and backend code. This can be seen by the number of commits in the GitHub repository. Thus, short release cycle was crucial for the successful completion of the project.

B. Requirement for a Distributed Development Model

The key aspect of ensuring project longevity is spreading out the responsibility for code review and integration across multiple team members. Especially as the project grows, sufficient resources are needed to cope with the changes and prevent working in "silos".

We implemented this by dividing the project into 2 parts, front end and backend. We each reviewed the code as it committed to the repository. That way we were all aware of the changes being made. We also used GitHub issues to communicate errors in the project. The issues were visible to the whole team and the person working on the feature can quickly pick up the issue and fix it.

C. Tools matter

For completion of any task, it is vital that we have the right tools. Linux Kernel Report explains how the switching from Bit-Bucket to GitHub changed the project trajectory over night.

The following are some of the tools we have used in our project:

1) Github: We used GitHub as the version control tool for our project. GitHub offers multiple features that helped in continuous integration. We were able to commit changes, pull changes made by others and report any issues. One feature that stood out was the GitHub actions. By adding build action to the project, we were able to ensure that the changes committed was not braking the project. Thus when we take the latest pull we will not have any issues in running the project.

2) Prettier: Prettier is an opinionated code formatter. It enforces a consistent style by parsing out code and re-printing it with its own rules that take the maximum line length into account, wrapping code when necessary thus increasing the code readability.

3) Pylint: Pylint is a static code analysis tool for the Python programming language. We used it to ensure correctness.

4) WhatsApp Messenger/Zoom: We used WhatsApp and Zoom for all communications related to the project.

D. Importance of a Consensus oriented model

Maintaining a Consensus oriented model ensures that all the team members are on the same page regarding the updated that are being made. It also states that a change will not be merged if a team member is not in agreement with it.

We implemented this using our WhatsApp group where we discussed the issues. One example of following a Consensus

model was regarding the status of the job applications. We provided inputs different application status. Once we were all in agreement on the various statuses, we continued with the development. It was important that be come to an agreement about this so the application can run as expected.

E. No internal boundaries in the project

This point emphasises the need for any developer to make a change to any part of the project if the change can be justified. As a result, problems are fixed where they originate rather than being worked around.Also, developers have a wider view of the project as a whole.

The rubric phrases this point as ”evidence that all team members can run code and configure tools”. During the project development, we agreed on the key features and divided them into tasks. Each member of the team was assigned a specific task. Every member of the team also agreed upon the tools that would be used to make working on the project fair game play. The WhatsApp group was highly active with progress updates and other communication which provided the team to work with zero internal boundaries.

IV. Conclusion

The Linux Best Practices was crucial in successful completion of the project. It gave us an introduction to how projects are developed in the corporate world as most companies follow these practices. We will definitely use and recommend these practices in our future projects.