Enigma

Sneha Madle
smadle@ncsu.edu
North Carolina State University
Raleigh, North Carolina, USA

Yugalee Patil
ypatil@ncsu.edu
North Carolina State University
Raleigh, North Carolina, USA

Saswat Priyadarshan
spriyad2@ncsu.edu
North Carolina State University
Raleigh, North Carolina, USA

Chirrag Nangia
cnangia@ncsu.edu
North Carolina State University
Raleigh, North Carolina, USA

ABSTRACT

This document focuses on different Linux Best Practices and how we implemented those in our project implementation. We collaborated with another group and enhanced their project by adding new features to the existing project and deploying the application on Azure.

ACM Reference Format:
Sneha Madle, Yugalee Patil, Saswat Priyadarshan, and Chirrag Nangia. 2022. Enigma. In *Proceedings of (CSC 510 Software Engineering).* ACM, New York, NY, USA, 2 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Our Project 2 focuses on enhancing a discord music bot that was capable of recommending songs based on user input and playing them on the discord voice channel. It was able to toggle music pause or play and play custom songs as well. We added new features like shuffling the songs in the queue, adding a new song to the queue, fetching songs from Youtube, and also deployed the application on Azure. In this paper, we will discuss our Project 2 rubric and how it relates to these 6 best development practices from Linux Kernel while giving examples as to how we applied these practices in our own development to make our team as efficient and communicative as we could.

CSC 510 Software Engineering, Oct 09,2022, Raleigh, NC

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

2 LINUX KERNEL BEST PRACTICES

The industry Standards for Software development enable developers to work in an organized and efficient manner while reducing dependency on other team members. One of the standards followed is the Linux Kernel Best Practises. We inculcate this standard in our project development to enhance and improve the project quality. In the following subsections, we describe how these best practices were adopted by our team.

2.1 Short Release Cycles

Short Release Cycles are important so that there are clear milestones to aim for and that the software as a whole is periodically being reviewed for what is getting added to it. This allows for constant evaluation of what is important, allows for flexibility in direction, and incorporates Agile methodology. It helped new code to be immediately integrated into a stable release.// During our implementation, it was important to have short releases as it helped us to identify issues and fix them efficiently. We implemented both the front-end and back-end simultaneously so it was important for us to have short releases to make sure the front-end and back-end are integrated and worked well. The releases were always performed from the main branch for a stable release. For every task and bug, an issue was created, picked up by a member, and resolved.

2.2 Distributed Development Model

The Distributed Development Model essentially focuses on task distribution between various team members to decrease the dependency of the team on each other and enable faster development of the software. For example for kernel development, the best practice followed was to assign different portions of the kernel to each team member. The portion assignment could be random, familiarity based or preference based.
We adopted this by following a similar work division. Initially, we divided the project tasks into adding new features

by each team member. Each member worked on adding a new feature like adding a new song to the queue, shuffling the song, fetching the song from Youtube, and deploying on Azure. And then we would have regular planning meetings to discuss the project's progress. Pull requests were reviewed by other team members and once we agreed on the feature development, the code was merged into the main branch.

2.3 Consensus Oriented Model

The Linux kernel community strictly adheres to the consensus-oriented model, which states that a proposed change cannot be integrated into the code base as long as a respected developer is opposed to it. This ensures that the kernel's code base remains as flexible and scalable as always. While implementing our project, we discussed different approaches and issues by making use of chat channels. Every issue was closed after having a discussion and only if everyone agreed on the solution. We also incorporated a CONTRIBUTING.md and CODE OF CONDUCT file in our repository which reflects the standards and ideology that our group agrees on.

2.4 The No-Regressions Rule

The No-regressions Rule implies backward compatibility of the software being released with its current version. That is any software that was working at a given state should work in the same/similar manner without causing outage/ software issues.

To ensure this, we were persistent in writing quality code that follows industry standards. We added code style checkers, code formatters, and Syntax Checker. We also added GitHub workflows to automate builds and automate sonar cloud analysis runs. Hence, in case of any build failures, we would get notified. We also insisted on writing test cases that cover the base cases, and all corner cases to ensure the new code was not breaking any previous functionality. We also manually tested the code changes by running the software and performing a quick check for any visible code break.

2.5 Zero Internal Boundaries

A developer in most cases makes changes to only certain parts of the code base. But this does not prevent him from making changes to other parts of the code base. It's valid as long as the changes are justifiable. This practice ensures that the problems are fixed right when they originate. It is not necessary to provide multiple workarounds. It also indirectly helps the developer to gain a global perspective on the code base rather than sticking to a specific repository. The rubric has three tuples that could be mapped to this category: evidence that the whole team is using the same tools: and everyone can get to all tools and files This indirectly translates the fact that every developer within the group knows each and every software or tool that is being used in the system. This ensures that everyone is aware of how the project actually works rather than just focusing on one aspect of the project. evidence that the whole team is using the same tools (e.g. config files in the repo, updated by lots of different people) This ensures consistency which makes the process of knowledge transfer, sharing, and discussion easier. evidence that the whole team is using the same tools (e.g. tutor can ask anyone to share screen, they demonstrate the system running on their computer) and evidence that the members of the team are working across multiple places in the code base This point provides a proof of the fact that the developers have indeed worked on different aspects of the code. We have done most of our coding part in Python and deployed the application on Azure so all of us are using the same tools. We all are pushing to the main branch in our repository which gives us the idea that all the contributors are important and we trust them completely to directly push into the main branch. All of us can run the same code in our system and can perform all the tasks from our end. The main config files in our repository are being continuously updated by all the members along with the cog files.

2.6 Best practice for Deployment

Making use of deployment slots helps in avoiding production issues and downtime. Hence, deploying the application to a staging environment, validating all changes, and doing tests will help make sure the application is ready to be deployed in production. Now swap the staging environment and production slots. Also, continuously deploying the code to the staging environment will allow us to test the deploy branch and track every code change or new feature added.

3 CONCLUSION

By Following the Linux kernel Best practices, we were able to deliver high-quality software while reducing dependency as a whole on the team. This gave us the flexibility to work at each team member's pace and collaborate efficiently.