Chat Defender Bot : Linux Kernel Best Practices

Bharath Katabathuni
bkataba@ncsu.edu

Bapiraju Vamsi Tadikonda
btadiko@ncsu.edu

Vinay Kumar Reddy Perolla
vperoll@ncsu.edu

Sai Sree Nalluru
snallur@ncsu.edu

Swimitha Reddy Buchannolla
sbuchan2@ncsu.edu

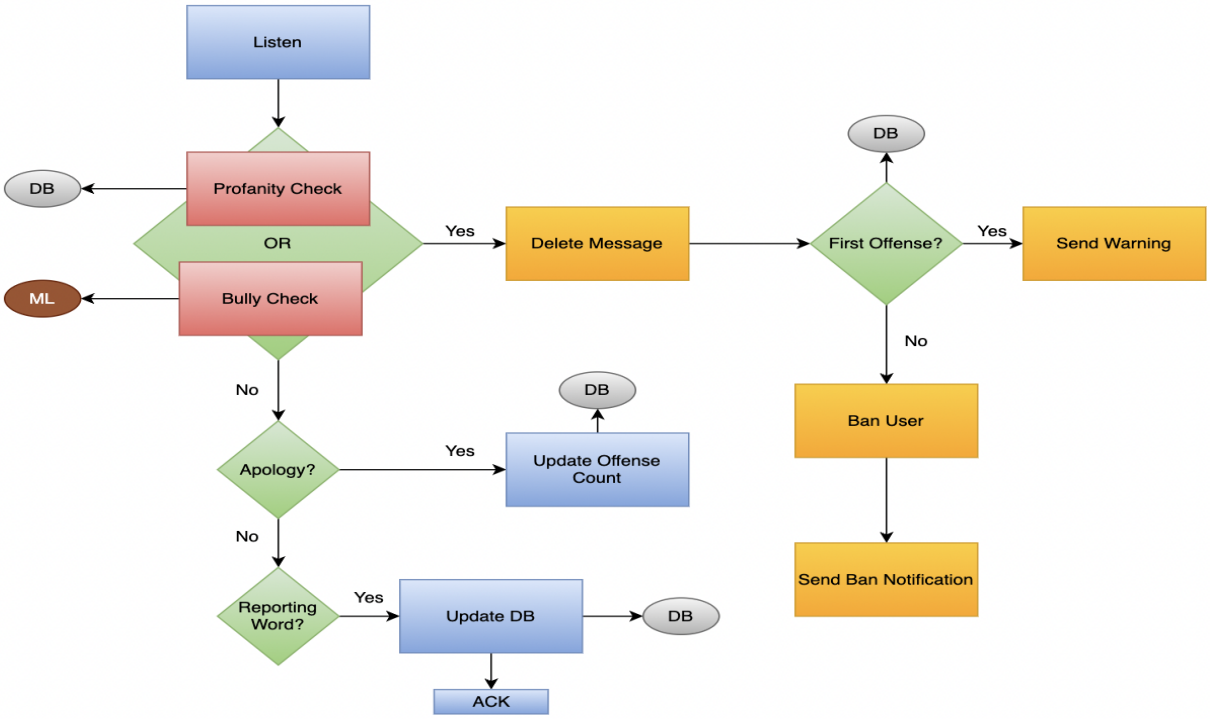

Figure 1: Project Workflow

ABSTRACT

There are many components which contribute to a successful project development. Following a standard rubric helps in streamlining the development process and delivering quality outputs. This article explores the Linux Kernel Best Practices, and comments on how they are incorporated in our project, Chat Defender Bot.

KEYWORDS

Linux Kernel, Software Development

1 INTRODUCTION

Chat Defender Bot is a Discord bot built to deal with cyber bullying and hate speech in chatrooms. It penalizes the users based on their past and current use of offensive words, and blocks them should the need arise. The members of the chatroom can report any proffanity that is not being picked up by the bot. This helps the model in learning, and improving its performance. The project is available publicly, hosted on Github [1].

The overall project development followed the kernel best practices as discussed below.

2 LINUX KERNEL BEST PRACTICES

Following are the best practices [2] adapted by us for the development of our project:

- Short Release Cycles
- Distributed Development Model
- Consensus Oriented Model
- No Regressions Rule
- No Internal Boundaries

2.1 Short Release Cycles

Having a considerably short gap between releases ensures that the new changes in code are brought to the users right away. Even

the smallest of changes can be incorporated smoothly. Instead of integrating huge volumes of code all at once, this practice allows the new changes to be added to the stable version when they are ready to be released.

Since the duration of this project itself is short, there were not many releases. However, this practice is still implemented through frequent code commits, whenever improvements are made to the code.

2.2 Distributed Development Model

Distributing the work among multiple contributors helps in dividing the workload, reduces pressure on the members and ensures that quality software is being delivered on time.

Our project development is well distributed with different tasks assigned to each team member. Changes are to be reviewed by at least one other team member before they can be pushed to the main branch. This ensures that the entire responsibility falls on a single person.

2.3 Consensus Oriented Model

Even though it might seem frustrating, no changes must be made without the approval of all the contributors. In case there are any disagreements, proper discussions should be held in order to address the issues, and achieve consensus.

Since it would be impractical to require a review from each person before merging, we made sure that anyone who has an issue with a certain feature could raise an issue, and the issue will be closed when all parties involved are satisfied.

2.4 No Regressions Rule

This rule states that if the current version of the software can be implemented in a given setting, all the subsequent releases, and improved versions must also work in similar settings. Successful implementation of this rule builds trust in its client base, and reassures them in their decision to keep using the product.

Our project ensures that all modules are implemented correctly in the current settings. Future developments to it can also be made in such a way that the existing functionality of the software is retained, ensuring the implementation of the no regressions rule.

2.5 No Internal Boundaries

Even though the project must be well distributed among the team members, there should still be no internal boundaries within the team to ensure that everyone can pitch in to help the others whenever appropriate after a proper discussion. While the distributed development model improves the efficiency of the team, it can also lead to delays in the development process at times, like when other team members' insights are required to solve an issue faster.

We made sure that if there are any justified changes a developer wants to make to a part of the code which is being worked on by a different member of the team, they can do so, even if they are not the person responsible for it, by raising an issue addressing the problem, and how they propose it could be resolved.

3 CONCLUSION

The Linux kernel is probably the most well-known and important open source project ever. In response to the variety of environments in which it is used, this kernel has a vibrant and active community that constantly helps the kernel evolve. As a result, there are more and more companies and developers involved in this. The development process has proven to scale up easily to higher speeds.
We followed these best practices when creating the chatbot. By utilizing the strategies we used, we were able to achieve the desired milestone in accordance with the team's objectives.

4 REFERENCES

(1) https://github.com/vamsitadikonda/chat-defender-bot
(2) https://go.pardot.com/l/6342/2017−10−24/3$xr3f2$/6342/188781/$Publication_LinuxKernelReport_2$017.pdf