# WalletBuddy: Linux Kernel Best Practices

Saksham Pandey
spandey5@gmail.com

Nihar Rao
nsrao@ncsu.edu

Palash Jhamb
pjhamb@ncsu.edu

Shruti Verma
sverma5@ncsu.edu

Manish Shinde
msshinde@ncsu.edu

## ABSTRACT

Software Engineering is continuously evolving and has undergone a significant change in terms of timeless principles and aging practices.[3] Development of new software is an arduous task and these principles are an essential aspect of ensuring the end product is of the highest quality. One such highly acclaimed rubric that is widely followed are the Linux Kernel Best Practices which have also been followed in our project - WalletBuddy.[2] This paper aims to provide a walkthrough of each Linux Kernel best practice and it's associated implementation in various phases of our project.

## KEYWORDS

Open Source, Kernel, Linux, Release, Software Engineering

## 1 INTRODUCTION

Software is a program or a set of programs containing instructions that provide desired functionality. Software Engineering is the process of designing and building something that serves a particular purpose and finds a cost-effective solution to problems. It is also working within well-defined and generally accepted human endeavor and WalletBuddy is our attempt with respect to that. WalletBuddy is a Telegram Bot that helps users track their expenses with just simple commands. It is written in Python and is hosted on GitHub. The project has been developed as a solution which follows the Linux Kernel Best Practices discussed below.

## 2 SHORT RELEASE CYCLES

short release cycles[1] refers to the integration of newly developed/fixed functionality into an already existing stable release in shorter intervals of time,rather than integrating long pieces of code all at once after a very long period of time. This concept has numerous advantages such as making new functionality/improvements available to users faster, maintaining a stable system without any major deviations/disruptions, increases team flexibility etc. We have adapted this practise of short release cycles in our project by committing and integrating our respective functionalities/improvements as and when they were reviewed as stable. This allowed us to implement new functionality and improvise on existing ones without causing any major disruptions to the existing stable version. This best practice also has major scope to be adapted in the roadmap of our project. Future members can integrate new code as and when it is stable, thus embracing short release cycles as a key best practice.

## 3 DISTRIBUTED DEVELOPMENT MODEL

Distributed development[1] is a software development model where different portions of the project are assigned to different individuals, based on their familiarity with the area. This helps in seamless integration and code review of the project. In WalletBuddy[2], we have adopted this practice by creating a pull request for the developed functionality and requesting a review from the other developer who is well-versed with the code and is aware of the functionality. By doing so, we ensure that the developer has followed standard coding practices. It also helps in identifying any missing checks like exception handling if not implemented by the developer

A significant takeaway from this practice is that one person is not burdened with the responsibility of the stability of the entire project. Sharing the load will improve the efficiency of the development and will help deliver the requirements on time with industry accepted standards.

## 4 CONSENSUS-ORIENTED MODEL

The Linux Kernel Community strictly adheres to the Consensus-Oriented Model which states that no change shall be implemented if even a single developer opposes it.[1] In projects where a large number of people are involved and directly hold a stake as a developer, exchange of different ideas is evident. It is important to have every stakeholder on board before implementing one of such ideas. This thought-process was seriously followed during the implementation of our project. Every developer contributed to the brainstorming process of the features even though they weren't writing the code for it. This ensured the consensus was maintained and every developer was aware of and agreed to every minute details of the project.

## 5 NO REGRESSION RULE

The No Regression Rule states that if a given program works in a particular setting, all ensuing programs will work there, as well. On the off chance that the community discovers that a change caused a regression, they work rapidly to resolve the issue. The standard gives customers or clients affirmation that updates won't affect their system; thus, they will follow the program as it gets updated. For example, if a company keeps updating their application's UI, the users will need to get used to the new interface after every update. This can cause hindrance to the user and the company can lose customers. As per our roadmap, there is scope for many updates. These updates will be done in accordance with the No Regression Rule, such that no update will change the basic functionality and interface of the tool.

## 6 ZERO INTERNAL BOUNDARIES

For a successful execution of a given problem statement or software, it is essential to delegate and divide tasks amongst team members. While this may seem to boost efficiency and localisation of bugs, it could also stagnate the overall development process. For example, a piece of software that was assigned to a team member might be taking too long to debug thereby stalling the upcoming release. If another team member who might have faced a similar issue and solved it in the past were to examine this bug, this issue can be resolved quicker. Therefore, although multiple developers work on their assigned tasks, it is important to have zero internal boundaries within the team to ensure bugs can be fixed by any member who identifies them first. While developing our project, we embraced zero internal boundaries wherein each team member stepped in to fix potential issues as and when they saw it fit after a review with the others.

## 7 CONCLUSION

The Linux kernel has been around for over three decades in the open source sphere and continues to be extensively developed and used in several industries around the world.[1] With thousands of developers contributing to and updating the kernel, the importance of adhering to software development best practices is evident from it's unmatched success. We have endeavored to emulate these best practices during the development of WalletBuddy.[2] The approaches chosen were inline with the goals of the team and therefore helped us reach the targeted milestone effectively.

## REFERENCES

[1] https://medium.com/@Packt_Pub/linux-kernel-development-best-practices-11c1474704d6
[2] https://github.com/smanishs175/WalletBuddy
[3] https://medium.com/pito-s-blog/a-view-of-20th-and-21st-century- software-engineering-67227009ad82
[4] https://medium.com/designing-distributed-systems/paxos-a-distributed-consensus-algorithm-41946d5d7d9