Sells Made Easy

Rahul Gautam
rgautam3@ncsu.edu
North Carolina State University
Raleigh, North Carolina, USA

Shreya Buddaiahgari
sbuddai@ncsu.edu
North Carolina State University
Raleigh, North Carolina, USA

Srilekha Gudipati
sngudipa@ncsu.edu
North Carolina State University
Raleigh, North Carolina, USA

Chetana Chunduru
cchetan2@ncsu.edu
North Carolina State University
Raleigh, North Carolina, USA

Sarika Vishwanatham
svishwa2@ncsu.edu
North Carolina State University
Raleigh, North Carolina, USA

Shubham Dua
sdua2@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 designed a custom e-commerce site that can be run by any local business owner free of cost to sell the merchandise at a large scale. All you need to do is setup the project and use it to increase your product reach.

KEYWORDS

E-commerce site, Agile Development, Linux best practises

ACM Reference Format:
Rahul Gautam, Shreya Buddaiahgari, Srilekha Gudipati, Chetana Chunduru, Sarika Vishwanatham, and Shubham Dua. 2022. Sells Made Easy. In *Proceedings of (CSC 510)*. ACM, New York, NY, USA, 2 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

In today's customer centric economy, products tweaked to the customer needs and desires have the highest market value. The recent boom in social media advertisement, has led to huge increase in demand for customized merchandise from small/local businesses. And most of the times these local businesses cannot get registered to world wide e-commerce sites due to local supply of the product. This project focuses on designing a custom e-commerce site that will help these local business to increase their reach while keeping technology cost at minimum.

CSC 510, NCSU, 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

Earlier, major releases would come less frequently following waterfall model. This would lead to several glitches and problems in the development cycle. In this case, long release cycles meant that major amount of code had to be integrated at once, which proved to be rather inefficient. It would make it difficult to fix bugs and make releases faster. Hence, Short Release Cycles was introduced in the Linux Kernel Best Practices to help incorporate Agile methodology. It helped new code to be immediately integrated into a stable release. Plus, continually integrating new code allows for introducing fundamental changes into the code base without causing major disruptions.
During our implementation, it was important to have short releases as it helped us to identify issues and fix them efficiently. We implemented both front-end and back-end simultaneously so it was important for us to have short releases to make sure frontend and backend is integrated and worked well. The releases were always performed from the main branch for 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 dependency of the team on each other and enable faster development of the software. For example for kernel development, the best practise followed was to assign different

portions of 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 frontend and backend. The frontend tasks were taken up by two team members and the backend tasks were taken by other two team members. And then we would have regular planning meetings to discuss the project progress. For the frontend as well as backend, we had clear task distributions. Each team member would pick up a task, perform the development in feature branch, test the code, raise the pull request. This pull request was then reviewed by other team member in the frontend/backend sub parts respectively. Once we reached the midpoint of the project development, we switched the roles of the frontend developers to backend developers and vice versa.

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 reflect 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 it's 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 on writing quality code that follows industry standards. Add code style checkers, code formatters, 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 base case, 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

Developers generally work on specific parts of the kernel; however, this does not prevent them from making changes to any other part as long as the changes are justifiable. To maintain kernel stability, it is better to fix the problem whenever it originates rather than coming up with different workarounds. To ensure that we implemented this best practice in our project, we made sure that the team agrees on the tools that will be used for implementation to have stable code throughout. The tech stack that was used is also updated in the README files. All the updates made to main branch was reviewed before it was merged. So changes made to different parts of the code was reviewed to ensure that the change is justifiable.

3 CONCLUSION

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