# Stock Trading Platform: How Group 42 Follows the 5 Linux Kernel Best Practices

Saketh Vangala (svangal*) Mukunda Varma Pericherla (mperich*)

Pranavi Sharma Sanganabhatla (psangan*) Kalyan Karnati (kkarnat*)
Srihitha Reddy Kaalam (skaalam*)
*@ncsu.edu
North Carolina State University USA

## ABSTRACT

The following is a report based on how the Linux best practices are followed for Group 42 of the first project. Our project is Stock Trading Platform, a web application that establishes a bridge between the buyers and sellers of the stock. This platform enables purchase and selling of stocks available in the market.

## KEYWORDS

best practices, software engineering

## 1 SHORT RELEASE CYCLES

The first Linux Best Practice is having short release cycles. It makes it possible to quickly build new features and address product bugs. Users benefit greatly from this approach because the product they use is constantly improved. Additionally, this makes it easier for the developers to test their draft code and to be able to correct any changes or issues with the subsequent version release. This is a fantastic approach to test any new features that might be expanded if they receive positive customer feedback.

Our group follows this model and Stock Trading Platform This encompasses the overall goal of the project: providing a platform to to sell and purchase the stocks by users.



## 2 DISTRIBUTED DEVELOPMENT MODEL

No one is accountable under the distributed development paradigm for writing code and approving what is introduced to the repository. Instead, all team members share these responsibilities. This strategy has the advantage of preventing new features from becoming stuck on a single person managing everything and ensuring that all team members have a working understanding of what is in the codebase.

Several approaches are used by our team to handle this: 1) All team members are free to work on the issues, and 2) when someone develops code, it cannot be merged unless one of the team members reviews it and accepts the modifications. The code is then added to the main. An individual is chosen to serve as a reviewer as the best practice. We should include someone who has experience working with that framework when working on the implementation of an item with that framework.

The project issues and pull requests serve as proof of this. Additionally, under insights in the repository, we can view the cumulative contributions made to the codebase by each individual to the project.

## 3 CONSENSUS ORIENTED MODEL

No group of users should prevent another group of users from contributing, according to the consensus oriented model. This implies that developers as a whole have a similar attitude. Before being integrated into the primary system, all generated changes must have the core developers' approval.

Our team's first step in achieving this best practice was to decide on our objective, which was to give our users access to a stock trading platform. We then established the milestones necessary to accomplish this aim and got to work. To hold project meetings and brainstorm ideas for implementation, we used a Discord server. To coordinate the work, we set up a number of channels on the server.

We were able to implement this strategy on the development side thanks to the pull requests' requirement for reviewers. We would be asked to make more changes if the reviewer was not pleased with the change in order to produce the desired result.

The primary evidence of this is focused on the project issues, pull requests, and discussion in Discord.

## 4 THE NO-REGRESSION RULE

The No-Regression Rule focuses on retaining backwards compatibility while still making progress. Users ought to be able to continue using the system in its current state even after an upgrade to a newer version. Developers shouldn't implement portions of features in smaller systems that people might begin using before the primary feature is implemented.

We follow this rule by not implementing any pieces of features that are not yet finished. For instance, we had market timings where user can sell and buy during that period only but before that feature is developed, the platform shouldn't allow any users to trade.

The users might have started using such features and expecting them to be around in the future if we had tried to integrate some of this in the code base. Users would have been unhappy that they could no longer use the sub-features if this feature was later dropped and they were removed. Due to our lack of action, we avoided breaking the no-regression rule.

The system's architecture, the code itself, and pull requests are the primary sources of proof for this rule (nothing was removed).

## 5 ZERO INTERNAL BOUNDARIES

Anyone can make changes to any aspect of the project thanks to the Zero Internal Boundaries best practice. This makes it possible for any developer to implement issues and features (similar to that of the distributed model). Every developer should have equal access to the development tools in order to make this happen.

Our group carried out this in several ways: 1) All developers used a virtual environment to manage packages, and 2) all developers had access to the entire codebase and could contribute to any section of it. Since the repository owner, kalyan, had to set up access for third-party applications, not all team members had access to third-party tools like CodeCov and Zenodo. We can specify specific versions of the packages we used in our requirements.txt using the virtual environment.

The project setup instructions in our repository's README and a list of specific packages may be found in the requirements.txt as proof of this.