XpensAuditor: How Group 2 Follows the 5 Linux Kernel Best Practices

Mithila Reddy Tatigotla (mtatigo*)

Sahithi Ammana(sammana*)

Sai Pavan Yalla (vyalla*)

Vineeth Dasi (vdasi*)
Sunandini Medisetti (smedise*)
*@ncsu.edu
North Carolina State
University USA

ABSTRACT

The following is a report based on how the Linux best practices are followed for Group 2 of the first project. Our project is XpensAuditor, a mobile application that helps to track expenses. This platform helps the user to control costs and see what they're spending their money on and how much they're spending.

KEYWORDS

best practices, software engineering

1 SHORT RELEASE CYCLES

Short release cycles are the first Linux Best Practice. It makes it easier to develop new features more quickly and fix product bugs. This strategy greatly benefits users since the product they use is constantly upgraded. This also makes it simpler for the developers to test their draft code and to be able to fix any improvements or problems with the ensuing version release. This is a great way to test any new features that might grow if they receive excellent client feedback.

Our group follows this model, and XpensAuditor encompasses the overall goal of the project: providing an app that helps users to track their expenses.

2 DISTRIBUTED DEVELOPMENT MODEL

All team members are accountable for writing code and approving what is introduced to the repository, under the distributed development paradigm. This strategy has the advantage of preventing single person to manage new features 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 a member develops code and creates a pull request, it cannot be merged unless one of the team members reviews it and accepts the modifications. The code is then merged with the main code. Someone who has experience working with a framework should be included when working on the implementation of an item with that framework.

This can be seen in project issues and pull requests. Under the Insights section of the repository, we can view the contributions made by everyone to the codebase of the project.

3 CONSENSUS ORIENTED MODEL

According to the consensus-oriented concept, no group of users should restrict another group from contributing. This suggests that developers adopt a similar mindset. All created changes require the core developers' approval prior to being merged into the main system. Our team's first step in achieving this best practice was to decide on our objective, which was to give our users access to the XpensAuditor app. 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 google chat. To coordinate the work, we discussed over chat and met in person also.

The pull requests' demand for reviewers enabled us to put this technique into practice on the development side. If the reviewer wasn't satisfied with the change's ability to achieve the desired outcome, we would be requested to make more adjustments.

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

4 THE NO-REGRESSION RULE

The No-Regression Rule focuses on retaining backwards compatibility while still making progress. Even after upgrading to a newer version of the application, users should be able to continue using it in its current version. Developers should not implement partial features that users might begin using before the primary feature is implemented.

We follow this rule by not releasing any features that are not yet completely implemented. For example, the SMS reader reads the transactions from the user's inbox and shows them in the app. Before this functionality was added, deletion and editing of the transactions was not made available to the user.

The users might have started using such features and expect them to be available if we had tried to integrate some of this in the code base. Users would be disappointed 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 project structure and pull requests are the proof that this rule was followed.

5 ZERO INTERNAL BOUNDARIES

The Zero Internal Boundaries best practice allows anyone to modify any component of the project. Any developer can now implement bugs and features thanks to this (like that of the distributed model). To achieve this, all developers need to have equal access to development tools.

Our group carried out this in several ways: 1) All developers used firebase to manage data, and 2) all developers had access to the entire codebase and could contribute to any section of it. Since the repository owner, Sai Pavan, 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.

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