Linux Kernel Best Practices - Flood Detection

Nelluru, Dedeepya
North Carolina State University
Raleigh, USA
dnellur@ncsu.edu

Chirumamilla, Raviteja
North Carolina State University
Raleigh, USA
rchirum@ncsu.edu

Vengali, Sai Kumar
North Carolina State University
Raleigh, USA
svengal@ncsu.edu

Immidisetti, Ratan
North Carolina State University
Raleigh, USA
rimmidi@ncsu.edu

Kanamarlapudi, Venkata Gnana Vardhani
North Carolina State University
Raleigh, USA
vkanama@ncsu.edu

ABSTRACT

The criteria listed by the rubric will be compared to the Linux Kernel Best Practices. We have categorized the parameters and explain why each one fits into or is similar to one of the publicly listed Linux Kernel best practices.

1 SHORT RELEASE CYCLES

The group covered by short release cycles is that the project members are carrying out sufficient tasks so that everyone can get their task. In shorter release cycles, shorter time duration is required for the small pieces of code to develop.

The developers are not packed with the idea of integrating huge features with risk of losing stability of the projects.
The new code is immediately integrated into stable releases. The disruption is reduced which improves efficiency. In this the tasks are segregated into parts constantly kept committing files so that it is on part with the whole.

2 DISTRIBUTED DEVELOPMENT MODEL

In a centralised system where a person reviews and approves code changes where this person makes a quick development and implementation of the given code base.

In a distributed system an expert in that particular field handles the task individually. This stops the bugs from getting through the review process which ensures the kernel stability. The elements of the distributed model are:

- Every choice is taken by a united vote.
- Group discussions have a round robin speaking order. A moderator which is suitable for round robin is used and is rotated among the group.
- Packaging standard conforms to code and can be downloaded from a random standard package manager.
- Work load is spread among the whole group.

Everyone has a chance to contribute to the code base equally under the distributed development model. Additionally, this makes sure that upgrades happen throughout the entire system rather than just in one area.

The following points all refer to various methods and techniques that can be employed to keep track of all the work that is being done on a distributed system.

- The code is committed by all the members of the group.
- If any issues seen, are reported and assigned to the concerned member of the group. The raised issues are resolved with the supported documentation.

We have used the MIT license and have a number of different badges like DOI badge, code coverage badge, python version badge. We have generated the different documents in a proper format, and with all the relevant information added so that anyone who uses our project can use it directly without any hassle.

We have also added descriptions of all the commands in different files so that the users can have an idea on how to use these commands. These files also contains the video description of the working of the application and how one can use the commands. We have also added the document generators.

3 ZERO INTERNAL BOUNDARIES

Only certain parts of the code base are changed by the developers in most cases. The issues are corrected at the beginning of the process. Multiple workarounds are not needed to be provided. All the developers know about the tools and softwares that are being used in the system.

This makes sure that every person in the team is aware about every aspect rather than focusing on just one part of the project. this guarantees consistency which makes the process of data sharing and knowledge discussion much easier.

Every process in the project is known to the entire team. It is proof that the developers have worked on every aspect of the project. The majority of our work was done in Python using an IDE like visual studio code and jupyter notebook. Because everyone in our repository is pushing to the main branch, we have the impression that all of the contributors are significant, and we can fully rely on them to push directly into the main branch.

4 TOOLS MATTER

Each type of project wants a mandatory group of tools and resources which requires it to be used in unison for the system to provide quality of service. These tools assist in keeping track of all the work that is taking place in the system. It helps in a smooth on boarding and understanding for a new person.

The user doesn't need to check for syntax and version control manually in every file. The tasks which map with this ideology are the following: making Use of version control tools, maintaining an up-to-date requirements.txt file, making use of code formatters, using syntax checkers, test cases to improve the coverage of code.

We use github as our version control tool and incorporate requirements.txt file in our repository. The part of style checkers are handled by visual studio code.

We used codecov for code coverage. Numerous test cases exist and they get executed each time any push to the main branch.

5 CONSENSUS ORIENTED MODEL

The linux community strictly attaches to the rule that a code cannot be combined in the main unless experienced and developer regards with approval. One group cannot make unwarranted changes to the code base at the expense of other groups.

Such limits are inflicted on the kernel code base and it remains scalable and flexible throughout. The following tasks appear to resemble and perceive the practice:

- Multiple people do multiple edits to the files CONTIBUTING.MD and CODE OF CONDUCT.MD, this check up on that each team member's opinion is taken into the consideration in making a decision.
- CONTRIBUTING.md lists coding standards and it adds lots of tips on how to expand the system without actually damaging existing things.
- Each decision is taken by the entire team and stakeholders.
- The minimal amount of bugs slip through a peer review process and issues are discussed and resolved in the process which ultimately maintains the stability of the product.
- Channel chat exists and also is active for discussing all the issues.

6 LOW REGRESSION RULE

DEVELOPERS always attempt to improve the CODE but not at the cost of quality. This is The ideal reason to introduce the no regressions rule, on the basis of this regulation, if a specified kernel works in a precise setting, each subsequent kernel must be able to work in this setting too.

However, if the system gets unstable due to regression, the developers won't waste the time on addressing the issue and get back to its original state. The practice that could be mapped to this is that the released features are not subsequently removed.