CSC510 project 2 Rubric Comments

Aadil Tajani Aastha Singh Arpit Choudhary Dhruvish Patel Kaustubh Deshpande

This document will illustrate the connection between Linux Kernel best practices and the Items in the project rubric.

I. Short Release Cycles

Short release cycles are critical to keep the mainline of the code working according to expectations because they facilitate finding smaller bugs rather than having a big update to the software where one does not know where the error actually exists. Some of the categories from the rubric that app to this best practice are as follows. Worload being spread over whole team: shows parallelization of tasks and optimization of speed of releases. High number of commits show that-constant small updates are being made to the codebase to actually facilitate short lifecycle. Style checkers, code formatters, code coverage etc have inbuilt mechanisms to test for best practices which will carry over to short release times.

II. Distributed Development Model

A distributed development model is super critical to achieving best outcomes in large codebases. Parallelization of tasks leads to increased speed and efficiency. It also gets more eyes on potential problems and as a result leads to better issue and bug recognition. A number of categories in the rubric reflect this. Numbers of commits by different people showcase that many people are working on an issue. Workload being spread over the whole team is another category that illustrates this. Issues being closed by multiple people shows that the issues are getting diverted to the person with the most know-how of the topic. Issues being discussed before they are closed is another indication of distributed sourcing of knowledge.

III. Consensus-Oriented Model

This principle is put in place so that concensus is achieved with everyone before a change is implemented. If a single developer is opposed to it, it will not be pushed. such a Practice reduces errors. This is showcased by existance of chat channels as they are important for concensus to be reached. Issues being closed with comments on it can showcase this as well. It is also important that multiple people commit in the same regions of the codebase to show that everyone is aware of the issues being worked on.

IV. The No-Regressions Rule

It is important to upgrade the code base, but quality cannot be compromised along the way, This is why Regressions are to be kept to a minimum. Multiple commits on the same issue ensures that releases go out in small batches which are easier to get right versus big releases. Use of Version control tools makes it easier to implement a regression if we do need to. Style checkers make code more readable, and thus reducing code errors. Another big tool is code coverage, which makes sure that implemented code is thoroughly tested before being pushed to a production environment. Test cases existing and being routinely executed is also another important way to avoid regressions by checking for errors.

V. Zero Internal Boundaries

It is important that a developer does not stick to only one part of the code base to solve an issue and look at a holistic, system wide solution to the problem. This avoids silos and facilitates a better quality of code. This is showcased by issues being commented on by multiple people, so that we encourage cross collaboration and coming up with system wide solutions. We can also showcase this by implementing test cases for issues and fixing them.