SlackPoint

Mithil Dani
mdani@ncsu.edu
North Carolina State University
Raleigh, North Carolina, USA

Ritwik Vaidya
rvaidya2@ncsu.edu
North Carolina State University
Raleigh, North Carolina, USA

Neha Kale
nkale2@ncsu.edu
North Carolina State University
Raleigh, North Carolina, USA

Rishikesh Yelne
ryelne@ncsu.edu
North Carolina State University
Raleigh, North Carolina, USA

Vansh Mehta
vpmehta2@ncsu.edu
North Carolina State University
Raleigh, North Carolina, USA

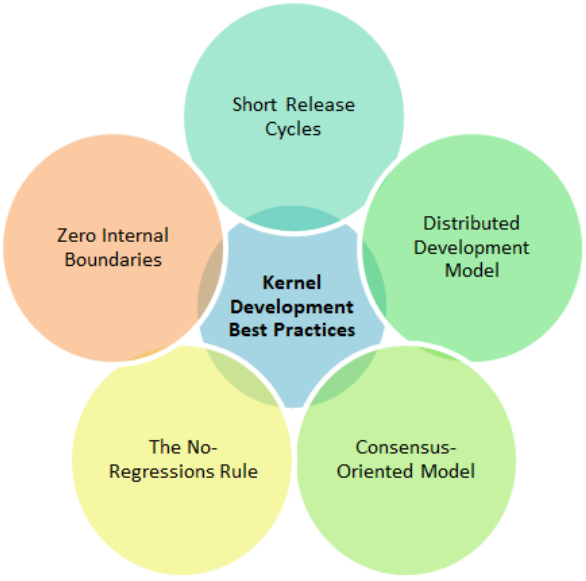

Figure 1: Linux Kernel Best Practices

ABSTRACT

Teams from all over the world use Slack, a messaging program for professional and organizational communication. SlackPoint is an integration for Slack which gamifies Slack. It creates an automated bot which allows users to add tasks, assign points to them and complete tasks. When tasks are completed the user receives points for the same. A leaderboard of the users of a workspace is also maintained. The rest of the document explains Linux Kernel Best Practices and how they have been utilized during the implementation of this project.

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/XXXXXXX.XXXXXXX

KEYWORDS

software engineering, messaging, automation, bot, gamification

ACM Reference Format:
Mithil Dani, Ritwik Vaidya, Neha Kale, Rishikesh Yelne, and Vansh Mehta. 2018. SlackPoint. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 2 pages. https://doi.org/XXXXXXX.XXXXXXX

1 DISTRIBUTED DEVELOPMENT MODEL

Following a distributed development model makes sure that everyone on the team is actively contributing to the code base and that no one person is in charge of all of the work. Additionally, it enables each team member to understand the various functionalities that are being used. We implemented this model in the following ways:
(1) We used GitHub as the version control system.
(2) The team members' tasks were distributed among them in accordance with their backgrounds and abilities.
(3) Issues were made that everyone could work on.
(4) Following a discussion with other team members, the issues were resolved. The 'Insights' tab of the GitHub repository, where

the closed issues may be seen, contains proof of this.
(5) Additionally, it is possible to view the commits made by the members, showing that everyone has contributed to the code base.

2 SHORT CYCLE RELEASE

In the past, major releases were made only every few years, which delayed the delivery of new features to clients and led to the merging of sizable portions of code into the code base. There are no or very few integration issues because to the regular incorporation of improvements and bug fixes made possible by short release cycles. Short release cycles are measured in the rubric. "Short release cycles," are made possible with the aid of version control techniques.

3 NO REGRESSION RULE

When a new feature or the existing code base are changed, regression testing is performed to prevent affecting existing functionality. Users are reassured by this that the application's current functionality won't change.
To guarantee that this is true, the following steps are in place:
(1) Using version control
(2) More than 50% of the code base is covered by the test cases.
(3) Whenever new changes are pushed to the code base, test cases are run.

4 ZERO INTERNAL BOUNDARIES

Flask, and PostgreSQL are the development tools used by all 5 members of our team, and they are all available to them. Additionally, we have a single public repository that contains all of the project's code. This means that everyone has access to modify the project as necessary, even though some contributors have made more contributions to the Flask development than others. All contributors have the ability to start the Flask server, install dependencies, and execute the application. This procedure makes problem troubleshooting simpler. For example, if an API call fails, any contributor can immediately identify the issue by looking at the relevant back end endpoint.

5 CONSENSUS ORIENTED MODEL

A consensus-oriented model requires that the majority of the project's contributors approve any changes. By doing this, it is possible to ensure that no part of the code base will be compromised when a new update is issued. Additionally, it aids in confirming the accuracy of the released code. Setting up meetings to discuss unresolved issues while the project is being developed is one of the finest practices. This conversation makes sure that the other developers are aware of the problem's solution and that the new code doesn't clash with any that has already been developed. To ensure the code is functioning properly, numerous tests are carefully run even after the new modifications are sent to GitHub. The criteria linked to these practices are as follows:
(1) There are numerous issue reports.
(2) Issues are closed.
(3) Before an issue is closed, it is discussed among all the contributors.