Mapping FindMyRoomie rubrics to Linux Best Practices

Saigirishwar Rohit Geddam
Arun Kumar Ramesh
Kiron Jayesh
Shandler Mason
Sai Krishna Teja Varma Manthena
sgeddam@ncsu.edu
arames25@ncsu.edu
kjayesh@ncsu.edu
samason4@ncsu.edu
smanthe@ncsu.edu
North Carolina State University
Raleigh, North Carolina, USA

ABSTRACT

FindMyRoomie is a Web Application provides a platform for North Carolina State University to find roommates of their preference. FindMyRoomie is a one-stop solution to your roommate finding needs. Our software has functionalities that allow you to filter and choose your ideal roommate. Moreover, the applications provides roommate suggestions based on user preferences Any NC State student could sign up with their NC State Email address from any corner of the world on our website and begin searching for roommates. In this report, we will be mapping the Linux Kernel best practices with the rubrics provided for the Project 1 of CSC 510 Software Engineering.

KEYWORDS

Software Development, Web Application, Roommates, NCSU, Python, Django, Linux Best Practices

ACM Reference Format:
Saigirishwar Rohit Geddam, Arun Kumar Ramesh, Kiron Jayesh, Shandler Mason, and Sai Krishna Teja Varma Manthena. . Mapping FindMyRoomie rubrics to Linux Best Practices. In *Proceedings of* . ACM, New York, NY, USA, 2 pages.

1 INTRODUCTION

The Linux Kernel Project is a humongous open source project which has been in use for decades. Its development methods, which have made it a great example of collaborative development, contribute to its effectiveness as a tool. Due to these procedures, Linux has been able to last for as long as it has without experiencing any quality compromises, which are common in projects of its size and duration. The Kernel Project upholds certain crucial guidelines which has allowed it to run efficiently. These guidelines include what we call as "Linux Kernel best Practices":

(1) Short Release Cycles
(2) Distributed Development Model
(3) Tools
(4) Consensus Oriented Model
(5) No Regression Rule
(6) No Internal Boundaries within the Project

In this report, we elaborate on these six best practices and map it to the rubrics of Project 1, while giving evidence as to how we applied these practices to our own project.

2 SHORT RELEASE CYCLES

Shorter release cycles ensure less frustration for the user and the developer. It reduces the chances of merging and pushing inefficient unstable code as shorter cycles ensure regular testing and update knowledge of the system architecture. This method also helps to bring fast and positive changes which result in the end user being satisfied. For these reasons, we integrate new code in short release cycles which are available as releases in our GitHub Repository. The following rubrics can be mapped to Short Release Cycles:

- Short Release Cycles
- Tests that can be run after your software has been built or deployed to show whether the build or deployment has been successful
- Issues are being closed
- Issues Reported: there are many
- Test Cases are routinely executed

3 DISTRIBUTED DEVELOPMENT MODEL

A Distributed Development model is the best way to develop any software. Code review and integration go smoothly with very little risk of a blow-up when different software functionalities are distributed to different people according to their expertise in the relevant field. The following rubrics can be mapped to this practice:

- Workload is spread all over the team
- Number of Commits: by different people
- Listing the important partners and collaborators on our website
- Do we accept contributions from people who are not part of your project?
- Do you have a contributions policy
- Is your contributions' policy publicly available?

- Evidence that the members of the team are working across multiple places in the code base

To uphold the distributed development model, the work has been equally distributed among the members of the group, the commits made to the repository through insights is the evidence for this. Furthermore, commits to different parts of the code base is another evidence that the whole team is using the same tools along with a well-articulated contribution policy that ensures that we accept contributions from people who are not part of the project.

4 TOOLS

Tools are a major advantage in developing vast projects with ease and efficiency. The following rubrics can be mapped to the usage of tools in our work:

- Use of Version Control Tools
- Use of style checkers
- Use of Code Formatters
- Use of Syntax Checkers
- Use of Code Coverage
- Other automated analysis Tools

In our work, we use Git Version Control tool to perform branching capabilities, ensure feature branch workflow, and collaborative work so that the version is stable and tested before release. Moreover, we use Black code formatter and Flake8 Style checker to ensure Python Standards are met. Also, we use Codecov to perform code coverage on our work and we use Unittest and CodeQl as other automated analysis tools.

5 CONSENSUS-ORIENTED MODEL

Integration to the code base need to be agreed upon by all and especially by people who have implemented some functionality and the new code block directly works with that. This makes sure that there are no changes made at the expense of any other feature or group, and the code remains flexible and scalable. The following rubrics of the project map to the Linux Best practices:

- Chat Channel: exists
- Issues are discussed before they are closed
- Project has an e-mail address or forum that is solely for supporting users.

To ensure consensus oriented model is followed, we have a chat channel on Discord along with GitHub Discussion where a particular feature or the need for any new feature are discussed. We make sure that a Pull request is raised from a feature branch to deal with the issues; two reviewers are assigned to the merge request to review, comment and discuss before the merge is accepted or closed. Once the merge is approved, the issue is discussed on what is expected and what is happening to meet the expectations and the issue is closed accordingly.

6 NO REGRESSIONS RULE

The No-regression rule is an important design decision as once the interface with the model gets pushed and is in public use, we should not alter that syntax. This ensures harmony in terms of user calls and less frustrations. We have ensured that we don't take away existing functionality but add to it. The following rubrics of the project map to the Linux Best practices:

- store documentation under revision control with source code.
- publish release history e.g. release data, version numbers, key features of each release etc. on web site or in documentation.
- Use of version control tools
- There is a branch of the repository that is always stable

The No regressions rule is upheld by documenting the releases on our documentations with versions by redirecting to our releases page on GitHub, along with having a stable "main" branch. Also, we ensure that there are no merge conflicts when merging a branch to the main along with the tests being passed successfully.

7 NO INTERNAL BOUNDARIES WITHIN THE PROJECT

The No Internal Boundaries rule ensures that the developer has an overall view of the project and access to the different parts of the project is available to every contributor. We understand that access to the entire view of the project is important. Even though individuals are working on different functionalities, it does not stop them from making changes in other parts of the code. This results in problems being solved at the source rather than having multiple paths to go through before making actual changes The following rubrics map to the no boundaries practice:

- Source code publicly available to download, either as a downloadable bundle or via access to a source code repository
- E-mails to our support e-mail ad- dress are received by more than one person
- whole team is using the same tools
- Project have a ticketing system to manage bug reports and feature requests