JobCruncher

Kartik Rawool
khrawool@ncsu.edu
North Carolina State University
Raleigh, NC, USA

Naveen Jayanna
njayann@ncsu.edu
North Carolina State University
Raleigh, NC, USA

Samarth Purushothaman
spurush@ncsu.edu
North Carolina State University
Raleigh, NC, USA

Tejas Prabhu
tprabhu2@ncsu.edu
North Carolina State University
Raleigh, NC, USA

Shubham Loya
ssloya@ncsu.edu
North Carolina State University
Raleigh, NC, USA

ABSTRACT

JobCruncher is an online job scraping and analysis tool that provides the user with the ability to filter jobs posted on LinkedIn based on the user's interest. LinkedIn is an employment-oriented online service that is a platform primarily used for professional networking and career development. This allows job seekers to post their CVs and employers to post jobs, hence a perfect site to scrap the job details from. This document aims to provide a walkthrough of each best practice and it's associated implementation in various phases of our project

KEYWORDS

scraper, LinkedIn, job search, job

ACM Reference Format:
Kartik Rawool, Naveen Jayanna, Samarth Purushothaman, Tejas Prabhu, and Shubham Loya. 2022. JobCruncher. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The Linux Kernel is a vast open source project which has been in existence for a long time and till today is supported and used by people all around the world. What makes a kernel which started on one computer to be used and developed further by many developers? We can learn a lot from the Linux Kernel practices on how to maintain a open source project. There are 5 Linux Kernel Best Practices defined as follows:

- Short Release Cycles
- Distributed Development Model
- Consensus-Oriented Model
- The No-Regressions Rule
- Zero Internal Boundaries

2 LINUX KERNEL BEST PRACTICES

2.1 Short Release Cycles

Shorter release cycles refers to adding small bits of code and releasing a stable version at shorter interval. It ensures that the code pushed is efficient and bug free. Also, it helps to pinpoint issues in the code quicker. Shorter release cycles helps to aim for small milestones at a time. Making short releases of few features is also a great way to get feedback for each change from the user. And make changes in the features at a stage when its easier to do i.e. at the early stage of development.

In the short period of development it was difficult to follow this practice. However, We tried to implement this practice into our project by frequently committing to the repository with changes so that the code is available to the team members. We created different branches for each feature and merged them once the feature was developed. The repository was setup with tests which ran on each pull request which ensured that only stable code.

2.2 Distributed Development Model

The Distributed Development Model breaks the project into parts and the parts are then assigned to different developers. This keeps in check that the responsibility is distributed and the entire load is not put on a single developer. Assigning different functionalities to different individuals based on their liking and expertise enables seamless code review and integration. The developer gets to learn new technology with the responsibility to only deliver a feature.

The Project 1 rubric supports this by including rubrics such as "workload is spread over the whole team", "evidence that the members of the team are working across multiple places in the code base" and "evidence that the whole team is using the same tools". The evidence for this can be seen in the number of commits and also the pull requests closed. Also, the commit history of each file shows that there are contributions from multiple members in each file. We distributed the work based on different parts of the project, each person created a branch and merged it once the part was completed and reviewed. All the members used the same tools and settings by maintaining a config file for the IDE.

2.3 Consensus-Oriented Model

There should consensus among the develops before the code is integrated into the stable version and released. The coding standards should be maintained for each integration. Also, the further development of the software should be agreed upon so that all the developers are working with the same goal in mind.

The rubric addresses this requirement in one major way: "Issues are discussed before they are closed". This includes the merging of pull requests, as we link our pull requests to our issues and a merge closes the issue. We made sure at least 1 reviewer check the code before merging. The rubric mentions another rubric "Chat channel: exists". For consensus to take place there must be a medium to communicate. We created a WhatsApp group to discuss further development, doubts in the feature and assignment of task. We conducted short meetings to discuss the progress made and in what direction based on consensus.

2.4 The No-Regressions Rule

The No Regressions Rule for Linux Kernel Best Development Practices means that as new updates come to a software, there is no removal of features that exist or the quality of code. We have ensured that we don't take away existing functionality but add to it. The rubric has points such as "Use of version control tools", " store your documentation under revision control with your source code", "publish your release history e.g. release data, version numbers, key features of each release etc. on your web site or in your documentation", "there is branch of repository that is always stable" This rubrics through documentation and version control can determine if the any feature are removed or the quality of code is reduced between the older iteration and newer iteration of the code base. To make sure we followed this rule, all features, bugs/bug fixes were documented in our issues and on our project board. This way, any changes can be quickly viewed to see if any significant reduction quality of code or features have been made.

2.5 Zero Internal Boundaries

The Zero Internal Boundaries rule states that there should be no boundaries between the contributors to a project as to who can access different parts of the project. Even though individuals are working on different functionalities, it does not stop them from making changes in other parts of the code. All the contributors should be able to use the same tool for development. This rule maps with the following rubrics "evidence that the whole team is using the same tools", evidence that the members of the team are working across multiple places in the code base" and "Source code publicly available to download". We have implemented Zero Internal Boundaries principle in the project as it can be seen that each of the team members contribute throughout various source files in the project, and we have used the same version of Python. We defined tools to be utilized for development at the beginning and kept a config file for the IDE so that all the members work on the same setup. We updated our requirements.txt file with all the libraries and modules required as we added new features.

3 CONCLUSION

The Linux Best Practices aligns perfectly with the Project 1 rubric. There were some practices which were difficult to apply to a project which was developed over a month. Developing the project, keeping these practices in mind, helped to understand the practices and also improved the coding acumen. These practices will be definitely utilized for the development of Project 2.