JobTracker

Shruti Marota
smarota@ncsu.edu
North Carolina State University
Raleigh, NC, USA

Landon Gaddy
rlgaddy@ncsu.edu
North Carolina State University
Raleigh, NC, USA

Li-Chia Chang
lchang2@ncsu.edu
North Carolina State University
Raleigh, NC, USA

Huangxing Chen
hchen63@ncsu.edu
North Carolina State University
Raleigh, NC, USA

Saminur Islam
sislam8@ncsu.edu
North Carolina State University
Raleigh, NC, USA

ABSTRACT

JobTracker is a personal Job board to add and track job applications in all their stages. This application allows users to provide information about jobs they have applied to, easily filter through applications and assign priorities to certain applications. JobTracker can also send reminder emails about applications with upcoming deadlines. Users can also use JobTracker's "Application Search" feature which allows users to search for job postings on LinkedIn and CareerOneStop within the program! The user can limit their search using keywords and JobTracker will display a list of associated jobs. The user can navigate to the original job posting, and easily add information about the application into JobTracker's system.

KEYWORDS

Track Applications, Job Information, LinkedIn

ACM Reference Format:
Shruti Marota, Landon Gaddy, Li-Chia Chang, Huangxing Chen, and Saminur Islam. 2022. JobTracker. 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 long lasting(26 years)and active project which provides a vital foundation for a large amount of academic studies and other relevant projects. It's so different from those slowly changed, obsolete projects that it's worthwhile to find out the reasons that make the successes of Linux Kernel be so long-lasting. Regarding this, there are 7 key aspects in general: Short release cycles, Distributed and hierarchical development model, Tools matter, Consensus-oriented model, No regressions, Corporate participation, No internal boundaries. For the purpose of getting a deep understanding of these, the article will analyse each of them based on the course project and its rubric.

2 SHORT RELEASE CIRCLE

A long release circle usually means the delay of features and updates, which leads to bad user experience and may lead to the elimination of products in competition with other products. To prevent this from happening, many product development methodology are recommending short release circle. For example, the product release frequency of a company using Scrum framework is usually by seasons(each 3 months), or even less. Running like this, the emergent customer requirements can be fulfilled in a relatively short time. Besides the quick response on new features and requirements, short release circle also facilitates the integration of codes. In other words, each time the team doing integration, there will be relatively less codes to work on than the Long release circle's. And the engineers won't have to wait for long time if they missed the current release.

Regarding the rubric, it's really hard to judge the Release frequency of a course project. But there are still some parts that reflect this: the rubric values the number of issues and commits(the higher the better) of the repository. Assume that we make each issues(commits) as a Scrum ticket, we can estimate the frequency of release if we combined a certain number of tickets as one release just like the real-world development.

3 DISTRIBUTED AND HIERARCHICAL DEVELOPMENT MODEL

Distributed Development best practices are a software development model in which development teams spread the project among the team members which is often considered as the mini projects that are brought together for a final software buildout. Normally, it's very difficult to work on a project which has only a single module to work on. It's cumbersome and difficult to track all the work from different developers. So, we divided our whole projects into different modules to keep track of changes from every member and review each other's work before pushing or merging it to the main branch. These helped reduce the workload and increase our efficiency to achieve every deadline we set.

In our distributed development model, we practiced the actions mostly from the suggested approaches from the project 1 rubric.

3.1 Spreading workload among team members

So, the whole project has multiple features including project design, defining use cases, functionalities, back-end software development, front-end development, and maintaining different branches, unit testing, black box testing, making presentations, videos for marketing our project. In our team collaboration, we complete every task very efficient way by dividing each work with equal distribution. Each of the members participated in weekly team meetings to discuss work progress and remaining tasks to complete. Each of the team members was responsible for checking our progress of each of us before deciding to complete a module.

3.2 Creating and closing GitHub issues

In our team collaboration on a distributed model, one of the most important things was to create issues to keep track of the progress of each one. We created issues for newly assigned tasks, fixed bugs, detected bugs, and asked the other team member to validate the issue and return with comments after working on the issue. If it is resolved we maintain an explanation before closing the issue.

4 TOOLS MATTER

Utilizing the right tools is imperative for the success of a project. The development, deployment, and maintenance processes for a project can be significantly easier by using the correct tools.

4.1 Version control

JobTracker is hosted on GitHub, a platform for software development and version control. By using GitHub, one can easily track changes as projects continue to develop. This is especially important when working on a project with multiple collaborators. Using GitHub allows each team member to work concurrently, review commits/code, add comments, create issues, and assign tasks.

4.2 Testing

JobTracker has been vigorously tested using a combination of blackbox and source code tests. This ensures that the software is working as expected/required and maintains quality of the code. By testing, flaws and defects can be found and fixed faster, which enhances the user experience.

4.3 Documentation

JavaDoc is a tool that is used to generate documentation for Java projects. Developers can provide information about classes, methods, and fields. JobTracker utilizes JavaDoc to maintain information about the source code, which helps future developers understand the implementation.

4.4 Style checking, code formatting, syntax checking

JobTracker utilizes CheckStyle and SpotBugs. CheckStyle is a tool that helps programmers write better code by checking if source code is compliant with a set of standard rules. Additionally, SpotBugs performs static analysis on the source code to look for potential bugs. Using CheckStyle and SpotBugs helps maintain the quality of JobTracker's source code.

5 CONSENSUS-ORIENTED MODEL

For a group of people to functions effectively, making effective decisions is very important. Consensus Oriented Model brings people together and helps them making better decisions by using widespread agreements in groups before make decisions. During our in person meetings we always hear everyone's idea related to the project. Then we decided to choose by reaching a general agreement among all of us.

For our discussions, we used weekly in-person meeting where normally we discussed weekly updates and further improvements on the project. There is a discord channel for our team which used for our daily communication to discuss different technical issues or improvements on previously agreed ideas.

6 NO REGRESSIONS

The no regression rule states that if a given kernel works in a specific setting, it gives users assurance that upgrades will not break their system. It means that any upgrade to the code or system should not cause an error on applications. In our project, the system works locally, so any future update in the project will not influence functionalities and interface on the client-side. Also, we use pull-request and issues documentation to verify and previously check modules that are working just as they were prior to the changes. Since every modification can be traced on branches, we are able to make sure before merging back to the main branch, changes will not affect the functionality of the main branch in our project.

7 NO INTERNAL BOUNDARIES

The goal of the No Internal Boundaries is to prevent blockers in code development and to encourage the most direct fix or implementation instead of having to find workarounds for arbitrary boundaries in development. Our team followed this policy when developing JobTracker to ensure the development was as efficient as possible. Because of the way we assigned tasks, people naturally became more specialized with parts of the codebase, but anyone had access to the entire codebase. This helped as when we found a bug in the UI that was caused by the backend, the developer who was adding that UI feature was able to fix the bug in the backend.