UnitPriceHelper Development: Connection with the Linux Kernel Best Practices

Pinxiang Wang
North Carolina State University
Raleigh, NC, USA
pwang25@ncsu.edu

Jiayuan Huang
North Carolina State University
Raleigh, NC, USA
jhuang52@ncsu.edu

Yuheng Zhu
North Carolina State University
Raleigh, NC, USA
yzhu63@ncsu.edu

Mengzhe Wang
North Carolina State University
Raleigh, NC, USA
mwang39@ncsu.edu

Yiran Zhu
North Carolina State University
Raleigh, NC, USA
yzhu59@ncsu.edu

ABSTRACT

UnitPriceHelper is a Google Chrome extension that allows to display the prices per unit for the products on multiple shopping sites, including Harris Teeter, Costco and etc. As a course project, this project has been self-assessed by the developers. This article is a comparation between this project's assessment rubric and five Linux Kernel Best Practices for software development, thus summarizing and reflecting on our development process.

KEYWORDS

Software Development, Linux Kernel Best Practices

1 INTRODUCTION

UnitPriceHelper is a CSC 510 course project, which is a Google Chrome extension that displays unit price of products on the shopping sites. Except for the project itself, we also paid much attention to its development process, which has been evaluated by a rubric[1]. In this article, we will compare our project's rubric with the five Linux Kernel Best Practices[2], which are

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

Through the comparation, we will point out what we can do to improve our development process in the future.

2 LINUX KERNEL BEST PRACTICES

2.1 Short Release Cycles

The Short Release Cycles principle means developers should always release the upgraded code in short time. Compared to long release cycles, short cycles have obvious advantages. Short release cycles can prevent the softwares from delays in technology and requirements, thus giving users better experience. To developers, short release cycles make the team more responsive to users' feedback and posible issues. For thess reasons, we upgraded this project at a fast pace.

Our first release implements the function of showing unit price on Harris Teeter's product list pages. The day after releasing v1, we began to work on the implementation on product detail pages. Then some team member proposed the idea of promoting this extension to multiple shopping sites, so we made it compatible to Costco, and are still trying more sites. These can be seen in the *Number of commits* and *Use of version control tools* rows in the rubric.

We did see the benefit of short release cycles during the development period. Since every website structure and every line of code was so fresh and clear in our mind, we don't need much time to recall or recheck the details when facing an issue reported, so we could upgrade it more efficiently.

2.2 Distributed Development Model

The essence of team work is collaboration. While deviding a project to several different part and spreading responsibilities to each member, a developer can get a more specific goal, thus concentrating him/herself on his/her area of expertise and keeping from meaningless distractions from irrelavant parts.

As shown in the rubric items of *Workload is spread over whole team* and *Number of commits*, my team did a good job in mission distribution. On the first days of starting this project, we held several meetings blueprint this project. Then the most experienced engineer in the team built a framework and assigned different functions to each members.

The *Issues reports: there are many* and *Issues are being closed* rows of rubric shows that we kept opening and closing issues while developing. To be more specific, we assigned issues to whom are responsible for them.

It was not a perfect temp, however. We met some incompatibility issues in the process. Two members wrote similar functions and commited them in the github repository without sufficient communication, so the project broke down for a while, as the issue #7 shown in our repository. But thing got correct simply by discussin and restoring.

2.3 Consensus-Oriented Model

When a team is working on a project, the team should make sure that they reach a consensus in each step thoughout the process. A healthy cooperation requires every member's participation in decision making and an agreement on the common goal.

My group believed in the importance of the consensus-oriented model, so we kept in close contact during the development period. The "chat channel: exists" item in the rubric shows that we had efficient online communication channels. We even stayed in the discord meeting room all day long when working, just to make sure that everyone can ask questions or discuss about features in the first time. Besides, we also booked study rooms in the hunt library to do face-to face strategic discussions in the early stage. Though there was argument, we reached consensus finally.

Our dedication to consensus-oriented model is also reflected in opening, discussing, closing issues, which is recorded in the rubric as *Issues reports: there are many*, *Issues are being closed*, and *Issues are discussed before they are closed*. When we open an issue, we would notify other team members to pay attention to it as soon as possible, thus resolving the issue quickly.

2.4 The No-Regressions Rule

No-regressions rule means that when developers upgrade the kernel, they have to make sure the upgrade won't cause any format of code degradation. If a given kernel works in a wpecific setting, all subsequent kernels should work there, too. This rule will benefit the users, thus building their trust in our products and making them long-term users.

What is shown this rule in our rubric is the item *Use of version control tools*. To use branches and merge, we tried our best to make each version stably upgraded.

On the other hand, we've noticed hidden meaning of this rule is that, if the team finds out there is a regression caused by some change unfortunately, they should try hard to address the issue as soon as possible. As shown in the *Issues reports: there are many*, there was a version incompatibility issue posted on Oct 4, which can be seen as a regression. We followed the rule, addressed the problem and fixed it within 5 hours. It gave us a profound warning, and since then we've became more careful to make sure there are no bugs before each commit.

2.5 Zero Internal Boundaries

The Zero Internal Boundaries principle means that there shouldn't be any boundaries when a developer need to use tools or functions in any part of the whole project. Though different members have different specialized fields and are responsible to different parts, they should have the accessibility to every part. This rule can help all developers build a global view of this project as a whole, thus following up better. Besides, this can effectively avoid duplication of methods and code.

Our github repository is the most direct manifestation of the absence of internal boundaries, which is listed in *Use of version control tools* in the project rubric. Everyone can access everything in this repo. Additionally, *Number of commits: by different people* also shows that sometimes some files are commited by different people.

3 CONCLUSION AND FUTURE WORK

UnitPriceHelper is the very first development project for some of the team members. In this practice, we not only implemented the practical functions in this Chrome extension, but also learned a lot how to standardize the development process. In conclusion, we almost completely followed the five Linux Kernel Best Practices. I'd like to regard this cooperation as a pretty remarkable practice and give my sincere thanks to all my team members.

However, there are several things we need to do better in the future. First of all is about the consensus oriented model. We did reach a consensus before each step, but they are some hasty consensus to some extent because we have to decide everything quickly due to the short time. More opinions should be considered carefully in the future decision making.

Secondly is about the no-regression rule. Section 2.4 wrote that we solved the incompability problem quickly, but in fact the process of finding the problem was painful. We should make a better use of version control tools next time to prevent this kind of problem in the first place.

REFERENCES

[1] Group 7. 2022. *Rubric for Group 7 Project 1.* https://github.com/yzhu27/UnitPriceHelper/blob/main/docs/rubrics.md
[2] The Linux Foundation. 2018. *2017 State of Linux Kernel Development.* https://www.linuxfoundation.org/2017-linux-kernel-report-landing-page/