# Teamformationassistant Mapping to Linux Kernel Practices

Yuchen Liang
North Carolina State University
Raleigh, NC, USA
yliang24@ncsu.edu

Gaolin Zhang
North Carolina State University
Raleigh, NC, USA
gzhang26@ncsu.edu

Yuyang Jiang
North Carolina State University
Raleigh, NC, USA
yjiang35@ncsu.edu

Minghao Guo
North Carolina State University
Raleigh, NC, USA
mguo6@ncsu.edu

Tianyi Cong
North Carolina State University
Raleigh, NC, USA
tcong@ncsu.edu

## ABSTRACT

This paper explains Linux Kernel Best Practices for Software Engineering and the benefits.We also discusses the connections between these and the items listed in the CSC510 Project 1 Rubric[1].

## KEYWORDS

software engineering, Linux Kernel

## 1 INTRODUCTION

The Linux kernel[1] is a free and open-source, monolithic, modular, multitasking, Unix-like operating system kernel. It was conceived and created in 1991 by Linus Torvalds for his i386-based PC, and it was soon adopted as the kernel for the GNU operating system, which was created as a free replacement for UNIX. Since then, it has spawned a large number of operating system distributions. Linux Kernel contains several rules and practices for software development and we concentrate on two of them. One is Short Release Cycles, the other is Distribute Development Model.

## 2 LINUX KERNEL DEVELOPMENT BEST PRACTICES

In Software Engineering Project 1, we will complain how Short Release Cycles and Distribute Development Model be related.

### 2.1 Short Release Cycles

The milestones are related to the Short Release Cycles which may separate the whole project clearly. It sets time when a piece of work in the project must be finished and the new features must be accomplished. Short Release Cycles separate the project accurately and offer the potential ability to achieve a huge project. At the beginning of this CSC 510 project, we make a decision to use the Short Release Cycles instead of releasing the full content all at once. There is no doubt it is crucial to plan every step for this one-month project. We pull the files from the GitHub frequently and push or merge the new files on time. We also organized several project meetings to discuss the schedule in the past one month periodically. For instance, the milestone of our project is totally clearly.

Hence, compared with the full releases, we chose this Short Release Cycles practice which help us to finish each trail function periodically.

### 2.2 Distributed Development Model

In group software development work, distributed development model is a good way to improve the efficiency by devide the whole project into several tasks. In our daily software development, some members of one team always do much more work than other members, which violates this model. So in the rubic of the project one of CSC 510, it is necessary for all teams to spread workload over the whole team. Also, the tasks arranged to any member of the team should fit their capacity and speciality.

In the process of our finishing project, we follow the rubric[2] and this model by several ways. For instance, members who do well in backend development are responsible for backend work by javascript while people doing well in frontend development do frontend work. Besides the tasks arrangement, communication is also an important part of this model. Thus, we communicate with each other nearly everyday by WeChat group to talk about the coding progress of everyone and meet in the library everyweek to help each other. What we do is also for the rubic that chat channel should exist.

### 2.3 CONSENSUS-ORIENTED MODEL

It is important for a team to have consensus no matter what they are doing. For our project, we didn't follow the consensus-oriented model at first, which lead to conflict of program we constructed. After that, we strictly follow this model by chatting frequently to reach the consensus of everything and using the same tool as the rubric[2] saying. In this way we worked more efficiently and avoided so many conflicts, which saved us much time.

## 3 CONCLUSION AND FUTURE WORK

After developing this project, we have further understand of The Linux Kernel best practices by trying to use some of them.

However, because we have so short time for this project and we haven't completely finished it (there may be other teams continuing to extend our project), there are still some rules and models we haven't tried like zero internal boundaries.

## REFERENCES

[1] https://en.wikipedia.org/wiki/Linuxkernel
[2] https://github.com/txt/se21/blob/master/docs/proj1rubric.md