Software Development Practices

Atharv Pandit
North Carolina State University
Raleigh, NC, USA
apandit3@ncsu.edu

Muhammad Ali Qureshi
North Carolina State University
Raleigh, NC, USA
mquresh@ncsu.edu

Shivangi Chopra
North Carolina State University
Raleigh, NC, USA
schopra4@ncsu.edu

Vaishnav Nagarajan
North Carolina State University
Raleigh, NC, USA
vnagara3@ncsu.edu

Vishwesh Sangarya
North Carolina State University
Raleigh, NC, USA
lsangar@ncsu.edu

ABSTRACT

This paper presents a brief overview of the practices in software development used for development of our software-AMATSA. Our work is inspired from the Linux Kernel Development Report of 2017.

KEYWORDS

software development practices, agile software

ACM Reference Format:
Atharv Pandit, Muhammad Ali Qureshi, Shivangi Chopra, Vaishnav Nagarajan, and Vishwesh Sangarya . 2018. Software Development Practices. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 2 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Asset Monitoring and Analytics Tool for sysadmins is a client-based solution for system administrators to monitor assets in their organization. This paper briefly discusses the characteristics of good software development practice we used for development of our software. Software development is challenging - programmers need to write clean, well documented code such that other programmers can understand what the code does. Collaboration between peers working on same part of the code needs to be effective to have less issues when integrating the feature. Software teams have to make customers happy by addressing their feature requests. Software teams increasingly have members who can work on any part of the project because of the flexibility to not move resources working elsewhere on the project.

2 DEVELOPMENT PRACTICES

In this report we elaborate on the five development practices and map it to the rubrics of our project, while giving evidence as to how we applied these practices to our development process. the development practices below are proven to work in a large project setting such as the Linux Kernel Development. The kernel project upholds certain crucial guidelines which has allowed it to run efficiently and are now called "Linux Kernel Best Practices"

2.1 Short Release Cycles

Software teams should embrace incremental releases rather than a major release. Incremental releases cause minimal disruption to production because there would be few new code. It is easy to recover from a regression issue or a new bug because we know the changes that broke the system. Incremental release cycles mean that every collaborator can have the latest stable code integrated into their feature branch. With this, we can avoid the pitfall of having to merge a new feature into an upcoming release and then find out there are integration issues. Incremental release cycles also help to keep the customers satisfied because they can see progress quickly. Developers are less worried about meeting deadlines to push code to release branches because if they miss the current release cycle, they can ship new code in the next release cycle within a couple of months. Such a setting can increase the productivity of developers too.

2.2 Distributed Development Model

Before the use of the Distributed Development Model, all code changes needed to be reviewed by a single person. As the code base grows, this becomes a cumbersome task for a single person. This gave rise to the Distributed Development Model where different portions of development-drivers, networks, etc were distributed among people based on their area of expertise. Through this, the project resources can cope with the increasing code changes and review in a software engineering project. We have used this in practice in the development of our software. We divided the code in as many small units as possible. Each team member was responsible for their unit. For example, one member was responsible for developing the driver code that integrates code from all other units. This leads to distribution of workload among team members. The fact that different people are committing to the repository shows that distributed development model is followed. Evidence that the team uses the same tools and styles checkers/code formatters and automated analysis tools also shows that the distributed development model is followed.

2.3 Zero Internal Boundaries

Having internal boundaries within software teams can result in delays and loss of time/effort with respect to the development or fixing an issue of a software component. Ensuring there are no internal boundaries helps speed up software development of a given component by allowing other developers using the said component to fix bugs or add features. This would assist them in continuing with their development cycle without having to wait for the software component to be modified by the developer working on that component. As a whole, by having no internal boundaries, and ensuring software changes are justified and well documented, it leads to quicker bug fixes and better release, as well as development time. When having no internal boundaries, it is important to justify any change made to a component by a developer who is not entirely responsible for the entire component. Code reviews and supporting tests for the components must be developed beforehand to ensure any changes to the component does not cause any breakages, conflicts and the primary functionality of the component is retained. Hence, by having zero internal boundaries within software teams, and by following good software development practices, the release time of the software and features associated with a component is shortened which gives way to efficient and effective software development.

2.4 The No Regression Rule

Due to short release cycles and incremental development approach, there exists a possibility for error.The new development, though working to introduce one new feature or functionality might affect the previously running code or alter the functionality.Thus, Regression defects are errors that occur as a result of changes made to code i.e. they're features that were previously working but are now broken thus hindering the quality of the built product. Since we need to have an incremental approach to our software development cycle, it gets extremely important to take care of regression after every cycle in order to maintain the code quality and thus leading to ease in understanding the root of errors since we now know and can narrow down that only the latest changes could have caused the said issue. In our development process, we took care of regression while following a distributed and incremental development model. Developing test modules for every functionality(unit -testing),having a strong code review process(by at-least two other team members) for every commit , automated running of test cases under git actions for every commit were majorly the steps taken to take care of regression in our development process. This helped us to develop a reliable and efficient software.

2.5 Consensus-Oriented Model

The decision-making procedure to be followed is one of the most significant aspects of a group activity. When a large group of individuals collaborates on a single project, it is critical that each individual participates in the decision-making process, that all suggestions from each individual are evaluated, and that the judgment is made accordingly. The consensus-oriented approach ensures that the choice that must be made has attained consensus, that is, that the majority of the members of the group must agree on the decision that must be made. We have followed the consensus-oriented model approach while developing our project. We made use of pull requests in our code which allowed members of the team to review the code written by other members of the team and discuss the changes and the code. We also made sure to acquire consensus before introducing any new features or expanding on current ones. This process enabled our group to come up with better solutions while still respecting the opinions of other team members.