Linux Kernel Best Practices - SE-37

Akhil Gangarpu Sudhakar
agangar@ncsu.edu
North Carolina State University
Raleigh, NC, USA

Jaydeep Patel
jpatel33@ncsu.edu
North Carolina State University
Raleigh, NC, USA

Shubham Waghe
swaghe@ncsu.edu
North Carolina State University
Raleigh, NC, USA

Venkata Sai Teja Malapati
vmalapa@ncsu.edu
North Carolina State University
Raleigh, NC, USA

Vijaya Durga Kona
vkona@ncsu.edu
North Carolina State University
Raleigh, NC, USA

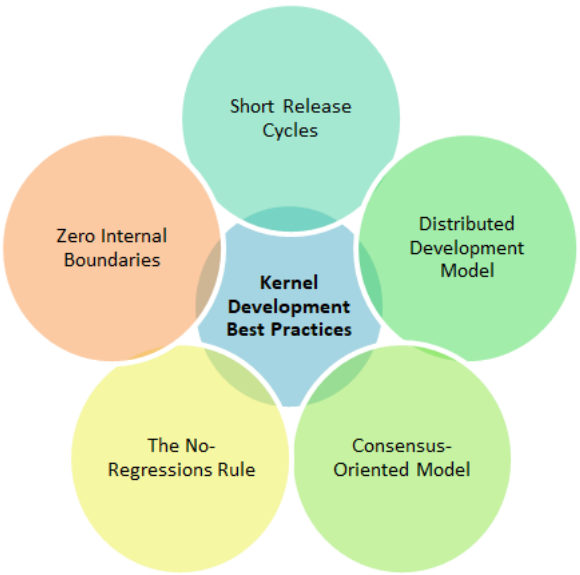

Figure 1: Linux Kernel Best Practices.

ABSTRACT

AI tools can generate many solutions, some human preference must be applied to determine which solution is relevant to the current project. One way to find those preferences is interactive search-based software engineering (iSBSE) where humans can influence the search process. Current iSBSE methods can lead to cognitive fatigue (when they overwhelm humans with too many overly elaborate questions. WHUN[8] is an iSBSE algorithm that avoids that problem. Due to its recursive clustering procedure,WHUN[8] only pesters humans for O(log2N) interactions. Further, each interaction is mediated via a feature selection procedure that reduces the number of asked questions. When compared to prior state-of-the-art iSBSE systems, WHUN[8] runs faster, asks fewer questions, and achieves better solutions that are within 0.1% of the best solutions seen in our sample space. More importantly, WHUN[8] scales to large problems (in our experiments, models with 1000 variables can be explored with half a dozen interactions where, each time, we ask only four questions). Accordingly, we recommend WHUN [8] as a baseline against which future iSBSE work should be compared.

This project has followed the below Linux Kernel Best Practices throughout the SDLC cycle.

1 SHORT RELEASE CYCLES

Short Release Cycles are important for the project development. This addresses lot of problems such as availability of new features and fixes without longer wait periods, minimal efforts for integration as code comes in short chunks rather than long chunks at once and reduces the premature code check-ins as the developer already knows the next release cycle even if they miss one. Longer Release

Unpublished working draft. Not for distribution.

SE Project, September 2021, Raleigh, NC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

cycles comes with multiple limitations such as disruption during integration and bugs. This project strictly follows short release cycles to ensure the smooth integration, testing and stable release. Following this practice helps in maintaining the quality of code as release go in short cycles. We are tracking all the Sprint releases through the JIRA[1] tool. This project is planned for 3 releases. Release version 1.0 is planned for Sprint 1 scheduled for September 30. Release version 2.0 is planned for Sprint 2 scheduled for October 15. Release version 3.0 is planned for Sprint 3 scheduled for October 30. Final release version 4.0 is planned for Sprint 3 scheduled for November month-end.

Our project rubric has the information on the number of commits which indicates that every developer is aware of progress of the work. As a developer in this team, every small change needs to be pushed to repository so that the other developers can start building the project on it. This helps in fixing any unexpected issues during integration of the project and identifying any bugs or dependencies to be added for the existing code.

2 DISTRIBUTED DEVELOPMENT MODEL

Distributed development model is a software development model in which each developer is assigned different modules to develop rather than centralizing most of the work with a single person. In centralized development model, absence of one single person affects the project and may impose greater dependency. These problems can be addressed by following the Distributed development model.

Our project rubric has specifications on the workload which is spread over the whole team (one team member is often X times more productive than the others and the number of commits: by different people). To address the same, this project uses the JIRA[1] tool to distribute the work where in user stories are created for every task that needs to be performed and assigned to developers, based on their choice and familiarity in the specific areas.. This helps in tracking the progress of the project constantly. Major project development work such as initial research paper survey, code modularizing, Static Analysis, Config, Packaging, and Documentation was categorized and created as an epic in the JIRA[1] tool, and subtasks are created under that to distribute the work among developers that were assigned to each developer. These subtasks are associated with each commit made in the repository so that we can trace back the Jira for more information regarding the changes.

This project uses a GitHub[2] repository to preserve and maintain the code where each developer can check in the code and every other developer has the access to see it. This enabled seamless code review and smooth integration. Trunk-based development is strictly followed in this project to reduce the complexity of merging and eliminate unnecessary divergence.

3 CONSENSUS ORIENTED MODEL

Consensus Oriented Model is the model where any proposed change cannot be integrated into code if any single respected developer opposes it. This can be frustrating to developers who find their code stuck for months and then needs to go through overhead of merging the code .But this ensures that the project remains suited to a wide ranges of users and problems. This improves the code quality and overall performance of the project . This project follows the Consensus Oriented Model throughout the SDLC cycle. This has been done by using “Planning Poker” where every developer of this project can speak through their votes for the proposals. Our project rubric has specifications such as the files CONTRIBUTING.md lists coding standards and lots of tips on how to extend the system without screwing things up and Chat channel exists. Microsoft Teams[5] is the platform for any discussion related to the project and this project also uses Quip[7] to share important project related data.

4 ZERO INTERNAL BOUNDARIES

Although developers are necessarily focused on their specific assigned parts of the project following the distributed development model, any developer can change any part of the project is a change is justified. This results in fixing multiple problems wherever they originate rather than waiting for the assigned developers. Our project gives access to all developers to change any part of the project if the change is justified. Every developer has the access to the GitHub[2] repository, where the code is located. Every individual in this project has access to all the tools used in the project - Jira[1], Quip[7], Microsoft teams[5], Research papers, Code base. Our project rubric has specifications such as evidence that the whole team is using the same tools: everyone can get to all tools and files (e.g. config files in the repository, updated by lots of different people, tutor can ask anyone to share screen, they demonstrate the system running on their computer) and evidence that the members of the team are working across multiple places in the code base.

5 NO-REGRESSION RULE

No-Regressions rule states that if a project works in a specific setting or environment, all subsequent projects should work there too. This project maintains a constant setting for developing and running the code to assure that code will not break during any upgrades. Our project rubric has the specifications such as use of style checkers, code formatter (AutoPep8[3]), syntax checkers (Pylint[6]), and config files in GitHub showing our config. This project has the Requirements.txt file where all the dependencies required for this project are added, and it can be run in any end system to get the required setting for this project to run. To ensure there won’t be any regressions in the future, we have chalked out a plan to add regression test suites.

6 TOOLS MATTER

We used various tools to develop and integrate the project. Source code was maintained on a GitHub[2] repository with regular commits, and versions. The planning and execution of the project was smoothly done using JIRA[1] which allowed us to create and assign tasks to each developer. To ensure code quality PyLint[6], a static anaylsis tool was used. All our code development was done on PyCharm[4] an IDE suitable for Python. AutoPep8[3] helped us in ensuring the code is formatted properly.

7 CORPORATE PARTICIPATION

This project has achieved corporate participation through ZOOM cloud meetings, Quip[7], Microsoft Teams[5].

ACKNOWLEDGMENTS

To Andre Lustosa and Dr. Tim Menzies for explaining the research and giving us the privilege to work on the same.

REFERENCES

[1] Atlassian [n. d.]. *Jira by Atlassian.* Atlassian. https://se2021-group37.atlassian.net/jira/projects.
[2] Github [n. d.]. *Github.* Github. https://github.com/ai-se/whun/tree/feature-se2021.
[3] https://pypi.org/ [n. d.]. *AutoPep8.* https://pypi.org/. https://pypi.org/project/autopep8/.
[4] JetBrains [n. d.]. *PyCharm.* JetBrains. https://www.jetbrains.com/pycharm/.
[5] Microsoft [n. d.]. *Microsoft Teams.* Microsoft. https://teams.microsoft.com.
[6] Python [n. d.]. *PyLint.* Python. https://pylint.org/.
[7] Quip [n. d.]. *Quip - collaboration tool.* Quip. https://quip.com/.
[8] Andre Lustosa Tim Menzies. [n. d.]. Preference Discovery in Large Product Lines. *https://arxiv.org/* ([n. d.]). https://arxiv.org/abs/2106.03792