A Comparison of Kernel Development Practices and Rubric Points

Gage Fringer, Stepan Kalinin, Brian Davis, Alex Carruth, Kunal Patange

September 29, 2021

1 Connections to Zero Internal Boundaries

When looking at the practice of Zero Internal Boundaries, the benefits come from its ability to ensure that all developers of a project have understanding of any aspect of a project that they may have interest in working on. Making sure that all developers have the ability to contribute to any area of the project allows for the ideas of the whole team to help drive the development of the software. In looking at the rubric, specific points that embody these are ensuring that members of the team are working across multiple places in the code base, everyone can get to all tools and files, and the inclusion of use-case tutorials. Ensuring that everyone has access to all of the files is one of the most important aspects to promoting Zero Internal Boundaries, and providing use-case tutorials allows for members who may not have as much experience in a certain section to gain a better understanding to be able to contribute. These two points then lead to members having the ability to work across multiple locations in the code base and avoid having developers siloed into certain areas.

2 Connections to The No-Regressions Rule

It is important both for developers and the user base of a software solution that the No-Regressions Rule is put in place, as this ensures that features of a product are kept in future releases, and also allows for developers to ensure that they are not making progress in the wrong direction. In looking at the rubric, specific points that are related to this rule would be the inclusion of test cases, making sure that these tests are routinely executed, and ensuring that static analysis tools are used during development. These three points are concrete ways for developers to ensure that the work they are doing keeps current software features maintained, while also providing metrics that can be conveyed to users to provide them a sort of guarantee of expected behavior for the software. Another point worth commenting on that isn't as prevalent as the aforementioned points would be in having developers make sure that a large proportion of issues on something like GitHub are related to failing tests. Providing thorough documentation of failures allows for the whole team to make sure that they understand the issues with features, and allows them to navigate developing a solution without sacrificing current standards set by the code.

3 Connections to the Consensus-Oriented Model

Consensus-Oriented decision making is important to the development process as it ensures the direction of development is agreed upon by the whole team. Without this level of communication, developers may run into problems in determining the scope of a project, or advancements to the code base may not meet full team expectations. A point from the rubric that would help to establish this sort of baseline would be to ensure that a chat channel exists. Without some form of real-time communication on a team, it would be increasingly difficult to manage aspects of a software solution as the code base grows. Many of the points for documentation would also be relative for the implementation of this model, as being able to create consistent documentation of things such as use cases or demonstrations of the code features would ensure that a development team has a consistent view of the product, and that decisions have been made with regards to the direction of the solution.

4 Connections to the Distributed Development Model

Being able to implement a Distributed Development Model for a software solution would ensure that developers are working on multiple aspects of a project, which would ensure that developers all have a stronger understanding of a product, as they should have all interacted with more aspects of it. Points from the rubric that are key examples of this model are making sure that commits are being made by different people in different areas of the code, having developers use the same tools for developing and updating code across the code base, and including files such as the CONTRIBUTING.md file to make sure that tips are provided to other developers to extend the system without introducing failures. All of these points work to assist a team in making sure that each of the members is equipped and able to develop in different areas of the project, allowing for the work to be split so as to avoid creating internal boundaries. The points mentioned are a better fit for this aspect of kernel development instead of Zero Internal Boundaries because these points are generally more concerned with a developer's ability to produce meaningful code in multiple areas by providing evidence and tools that demonstrate this, while Zero Internal Boundaries is more concerned with ensuring that developers are able to access the different aspects of a project.

5 Connections to Short Release Cycles

Short Release Cycles, which can be generally observed as following a more Agile-driven approach to development, allows for the direction of a software solution to evolve as the code base evolves, ensuring that the requirements of a solution are always kept as the central focus of development. Having shorter release cycles also ensures that developers split goals/features into smaller, more easily attainable goals during development to keep developers from getting stuck on large problems that were not properly accounted for. The inclusion of Short Release Cycles is something that is explicitly mentioned on the rubric, and showcases the importance of having a model such as this. Having a large number of issues and making sure that many of them are getting closed is another sign that goals are being broken into more manageable chunks, allowing for a better analysis of how fast problems are progressing. Ensuring that there are a large number of commits also helps a team to make sure that all members have each others most recent work at any given moment, allowing for more efficient development when everyone has recent code.