Rachel Theis

Module 6 Assignment

CSD 380

6.29.25

**Strangler Pattern at Blackboard (Kim et al., 2021)**

In 2010, Educational technology pioneer Blackboard found that they were running into challenges maintaining and scaling their flagship product that had code dating back to 1997. Blackboard was battling increasing complexity, greater lead times, and worsening customer experience. Research on this issue found that while lines of code continued to increase, the number of commits was decreasing, signaling the challenges the development team was facing when changing the monolithic repository.

Blackboard understood the importance of getting its legacy code under control and modernizing the repository to mitigate the challenges developers were facing. The company decided to implement a strangler pattern solution, which used “building blocks” that enabled developers to work outside of the monolith in a decoupled state. These building blocks were separate modules that were decoupled from the main program using an API. As building blocks were made available to developers, the size of the monolith gradually decreased, which was evidenced by shrinking code line numbers. The building block system was preferred over the monolith by developers and enabled them to work more efficiently in a safe environment that protected the global environment.

**Lessons Learned**

1. Monolithic architecture presents scalability and maintainability challenges.
2. Strangler Pattern Approaches can preserve customer experience while an application undergoes changes.
   1. Since Blackboard needed to modernize its flagship product, it was critical that users could still access this product with limited interruption. Taking an all-at-once approach could have put educators in a bind and caused a loss of revenue for Blackboard.
3. Feature branching can be a useful strategy when dealing with legacy code or other large-scale modernization or migration efforts to keep the core of the program safe during a volatile time
   1. While trunk-based development aligns well with DevOps principles like CI/CD, and is a great choice under the proper circumstances, taking a feature branch approach may be the right solution when projects aren’t running smoothly or are undergoing major changes.
4. When legacy code starts to be inefficient, it should be solved as quickly as possible to avoid taking on unnecessary technical debt and snowballing issues.
5. Code should be developed with modularity to make it more scalable, manage complexities, and enable developers to work more easily on it.
6. When projects are planned, teams should consider future scalability potential and choose an architecture accordingly.
   1. Monolithic architecture is still a relevant framework, but teams should weigh the benefits and challenges of each type of architecture to determine which is best for their project.
7. An organization should use quantitative data like lines of code vs. commits to determine how well a project is flowing and if a project is suffering from complexity or architecture issues.

**References**

Kim, G., Humble, J., Debois, P., Willis, J., & Forsgren, N. (2021). *The DevOps handbook : how to create world-class agility, reliability, and security in technology organizations* (Second Edition). IT Revolution.