Jeffrey Reid Module 6.2 February 9, 2025

Case Study: Strangler Pattern at Blackboard Learn (2011)

Blackboard is a great example of when to use the Strangler Fig Pattern. In 2011, the company started to see cracks in their software from their continued use of the J2EE code base. The aged Perl code had been present since 1997 and was beginning to take its toll. This was evident in the increased times required to test code, build it, and integrate it into the overall system. Things were taking longer and longer and were becoming more error prone as time went by. A great way to see this was a series of commit graphs that were tracked at the time between 2005 and 2011. In these graphs, we see that the number of commits to the repository for Blackboard become more and more infrequent as the years went by. Even basic feedback from the integrations took longer and longer, usually at least a full day. However, the overall quantity of code was on the rise. This spelled difficulty for each attempt at pushing commits. The complexity of the system was becoming too much bear with.

In order to solve this system-wide problem, it was decided to enact the Strangler Fig Pattern to change the entire architecture around. The goal was to change the cumbersome Monolithic structure they currently had into a more isolated Micro-services type. To do this, they built “Building Blocks” that allowed them to separate certain modules from the main hub. They used fixed API’s to connect to the old Monolithic structure when needed. The big benefit to this was that they were able to separate communication and coordination for different sections of the code base. Different services became independent and helped decrease the overall code complexity of the system.

The arrival of the Building Blocks helped to slowly decrease the code size of the repository. This was a result of transferring the old Monolithic code segments into the repositories created by the Building Blocks. Most developers at Blackboard at this time preferred to work with the newer repositories over the old thanks to the increased flexibility, safety, and autonomy. After this transition, we see from 2012 to 2013 that the number of code commits from the repositories began to spike upward exponentially. By having all of the modules separated, the teams were able to be significantly more productive. This resulted in safer work that had isolated errors, if any, that avoided the global error impact of the past. The process became simpler than before and so the amount of work that could be done increased substantially. Everything improved, from feedback quality and speed to overall code quality. The architectural transition thanks to the Strangler Fig Pattern was a huge success.

The lessons learned from this are as follows:

1. When communication (from feedback to simply working to push code) and productivity become a problem due to the complexity of the code, it may be time to consider changing architecture styles. This is especially true if you have a large application or program that has a lot of moving parts.
2. Always strive to work with code in a safe and productive manner. If you feel you need to make changes to the way you’re working, do it. Change is not a bad thing. Don’t put this off until it’s too late!
3. It may not always be possible for smaller applications but strive to be agile in your code base as early as you can. This way, if an architectural revamp is needed in the future, it’s easier to roll out.
4. I’m not sure how feasible this one is but if you envision your application to be large from the beginning, consider starting off with a Micro-services Architecture rather than a Monolithic one. The separated services may make things a bit complex in the beginning but in the long run, it’ll serve to benefit the entire system by isolating services as the application gets larger. The isolation of each service will improve independence, freedom, and help keep the code safe.

Citations

*Kim, G., Debois, P., Willis, J., Jez Humble, & Allspaw, J. (2021). The DevOps handbook : how to create world-class agility, reliability, & security in technology organizations. It Revolution Press, Llc.*