Derek Livermont

CSD 380

Module 6

Blackboard Inc. is a huge player in the education application space. I remember using the platform back in 2010 during my first college experience and now here at Bellevue. I actually vividly remember regular down times and outages of the application back then at the most inconvenient times. Reading the case study, it’s no surprise why.

Blackboard, like many companies finding fast success in a growing market like online education, worked to keep up with growth and the requests for new features. In the process, however, they didn’t take the necessary precautions to prepare themselves for growth like decoupling the individual processes that made up the full application. As a result, they became tied down and limited by slow testing and pushes to production that had the possibility of taking everything down.

The graph in the book comparing the lines of code to the number of code commits is extremely telling. What better way to tell the story of an institution where developers don’t feel safe committing code than to see it represented. How can a company continue to innovate when people don’t feel comfortable and empowered to do so?

Ashman’s idea to separate out functionality using the strangler fig pattern alleviated that stress. By creating the Building Blocks repository that separated out functionality and reduced risk as well as complexity for developers, he gave them back freedom and autonomy previously strangled by monolithic architecture. As a result, we see exponential growth in code commits in the new Building Blocks repository. Even when developers made mistakes, the results were smaller in scale and much easier to correct than global outages.

There are a few lessons to learn from this case. The first being is to not find yourself in Blackboard’s position in the first place. When prioritizing features and commits, it’s easy to lose sight of the overall architecture of your code base. There are a few situations where a tightly coupled architecture makes sense but, more often than not, it’s better to build a loosely coupled application from the start. This allows the flexibility of the “building blocks” structure that Ashman had to implement down the line. And that leads to the final lesson: time spent tackling technical debt and architecture is time well spent. Taking the time to analyze and find different ways of doing things might feel like taking time away from delivering features but it will pay down the line. Now that Blackboard has restructured, they can keep working on providing new features much more efficiently than before.

# Sources

Gupta, S. (2023, September 9). Decoupled Architecture & Microservices - Saurabh Gupta - Medium. *Medium*. https://medium.com/@saurabh.engg.it/decoupled-architecture-microservices-29f7b201bd87

*Decoupling Architecture*. (n.d.). https://www.itarch.info/2020/05/decoupling-architecture.html