Robert Dylan Bean

4/20/2025

Assignment 6.2

Professor John Woods

**Case Study: Strangler Pattern at Blackboard Learn (2011)**

This case study looks at Blackboard Inc. back around 2011. They're a big company making educational software, and their main product was called Learn. The problem was that Learn was built on really old code, going way back to 1997, even having bits of old Perl code mixed in. David Ashman, the chief architect, noticed this was causing major issues.

The main point the case study makes is that this old, single chunk of code (called a monolith) was getting really complex. It made everything slow down. Building the software, putting it together, and testing it took longer and longer and had more mistakes. It became such an issue that just getting feedback on whether a change worked could take a day or even a day and a half. This wasn't good for their customers.

Ashman saw proof of this problem in graphs showing their coding history. Even though the total amount of code kept growing, the number of actual code changes (commits) started going down. This showed him that developers were finding it harder and harder to actually make changes to the system. He knew they had to do something different, or things would just keep getting worse.

So, in 2012, Ashman started a project to fix this using something called the "strangler fig pattern." The idea here is to slowly replace parts of the old system with new, separate pieces, kind of like how a strangler fig vine grows over an old tree. What they did specifically was create "Building Blocks." These were like separate mini-programs or modules that developers could work on. These Building Blocks connected to the main old system through specific connection points (APIs), but they were mostly independent.

This change had big effects. Developers really liked working on the Building Blocks because they could work more freely and didn't have to constantly coordinate with everyone else. They had more independence. As a result, developers started moving code out of the old, clunky system and into these new modules. You could actually see the size of the old codebase start to shrink.

More importantly, developer productivity shot up. The graphs showed that the number of code changes being made in the new Building Blocks system grew really fast. Working in these smaller, separate modules was also safer. If a developer made a mistake, it usually only affected their small module, not the entire Learn system. This meant small problems instead of huge disasters. They also got feedback on their work much faster, which helped make the final product better.

The main lessons learned from Blackboard's experience are:

* Really old, large codebases (monoliths) become hard to work with, slow down development, and hurt productivity over time.
* Breaking down a big system into smaller, independent pieces (like the Building Blocks) using techniques like the strangler fig pattern is a good way to deal with this without having to rewrite everything at once.
* Giving development teams more independence (autonomy) to work on smaller parts makes them faster and more productive.
* Making changes safer (so mistakes have less impact) and providing faster feedback also helps improve speed and quality.

References:

Kim, G., Humble, J., Debois, P., Willis, J., & Allspaw, J. (2016). *The DevOps handbook: How to create world-class agility, reliability, & security in technology organizations* (2nd ed.). IT Revolution Press.