Clay Lankford  
Module 6.2  
06/28/2024

In 2011, Blackboard Inc., a pioneer in educational technology with a revenue of approximately $650 million, faced significant challenges with their flagship Learn product due to an outdated J2EE codebase dating back to 1997. Chief Architect David Ashman observed that the growing complexity and extended lead times associated with the old system were impacting developer productivity. The build, integration, and testing processes had become increasingly complex and error-prone, with feedback cycles taking up to thirty-six hours. Graphs from their source code repository revealed a decrease in code commits despite the increasing lines of code, highlighting the difficulty of introducing changes to the monolithic codebase.

In response, Ashman initiated a code re-architecting project in 2012, employing the Strangler Fig Pattern. This involved creating modular components known as Building Blocks, which allowed developers to work independently through fixed APIs, reducing the need for constant communication and coordination. As a result, the monolithic codebase began to shrink, and developers preferred working in the new Building Block modules, enjoying greater autonomy and safety. The transition led to an exponential increase in both the number of lines of code and code commits in the Building Blocks repositories, improving productivity and reducing the risk of widespread system failures.

Ultimately, Ashman concluded that the modular Building Blocks architecture significantly improved code modularity and developer independence. Combined with updates to the build process, this approach provided faster feedback and improved code quality. Utilizing the Strangler Fig Pattern and creating a modular codebase allowed Blackboard's teams to work more autonomously, tackling individual problems more safely and efficiently.

The lessons learned from this case study are clear. Legacy systems can severely hinder productivity and increase complexity over time. Implementing the Strangler Fig Pattern to gradually replace parts of a monolithic system with modular components can greatly enhance developer autonomy and reduce overall system complexity. Breaking down a monolithic codebase into smaller, independent modules allows for safer and more efficient development. Additionally, enabling developers to work independently within a modular framework leads to higher productivity and better quality code. Finally, improving build processes and modularity results in faster feedback and improved code quality, demonstrating the effectiveness of modular architecture in modern software development.

Resources:  
The DevOps Handbook: 2nd Ed. Gene Kim; Jez Humble; Patrick Debois; John Willis; Nicole Forsgren