**Module 6: Continuous Integration and Low-Risk Releases**

**Assignment: 6.2 - Case Study: Strangler Pattern at Blackboard Learn (2011)**

**Name: Arun Sharma**

**Course: CSD380-H326 DevOps**

**GitHub Link:** [**https://github.com/SharmaArun017/csd-380**](https://github.com/SharmaArun017/csd-380)

**Date: 02/09/2025**

= For this week's **Module 6 assignment**, we have to study the **Strangler Pattern at Blackboard Learn (2011)** and analyze how this approach was used to modernize a **legacy system** without disrupting operations. Blackboard Inc., a leading provider of educational technology, faced significant challenges due to an aging **J2EE-based monolithic codebase**, which resulted in **long deployment cycles, high complexity, and declining developer productivity**. To address these issues, the company implemented the **Strangler Fig Pattern**, a technique that allows for **incremental system upgrades by gradually replacing outdated components with modern modular structures**. This case study provides valuable insights into how the **Strangler Fig Pattern enables seamless migration, improves development efficiency, and enhances software stability**, making it a crucial methodology in **DevOps and software modernization strategies**.

In 2011, Blackboard Inc., a leading provider of educational technology, faced significant challenges with its flagship product, Blackboard Learn. The platform's legacy codebase, originating in 1997, had become increasingly complex and cumbersome, hindering development efficiency and product quality. To address these issues, Blackboard adopted the **Strangler Fig Pattern**, a software design strategy aimed at incrementally replacing legacy systems with new functionalities. This approach allowed Blackboard to modernize its application architecture without the risks associated with a complete system overhaul.

**Challenges with the Legacy System**

By 2010, Blackboard Learn's development team was grappling with the daily consequences of an outdated J2EE codebase. David Ashman, the chief architect, noted the presence of remnants from earlier technologies, stating, "We still have fragments of Perl code still embedded throughout our codebase." The monolithic nature of the application led to prolonged build, integration, and testing processes, with feedback loops extending up to thirty-six hours. This latency not only slowed development but also adversely affected customer satisfaction due to delayed updates and bug fixes.

An analysis of the source code repository from 2005 to 2010 revealed a troubling trend: while the codebase size steadily increased, the frequency of code commits decreased. This indicated growing difficulty in implementing changes, as developers had to navigate an increasingly intricate and interdependent system. Ashman observed, "The larger the product got, the longer our lead times and the worse the outcomes for our customers."

**Implementing the Strangler Fig Pattern**

To mitigate these challenges, Blackboard embraced the Strangler Fig Pattern in 2012. This strategy involves gradually replacing parts of a legacy system by building new functionalities around it, eventually allowing the old system to be "strangled" and phased out. Blackboard introduced modular components known as "Building Blocks," which were decoupled from the main codebase and accessed through fixed APIs. This modularity enabled developers to work more autonomously, reducing the need for constant communication and coordination across teams.

The transition to Building Blocks led to a noticeable decrease in the monolithic codebase's size, as developers migrated existing functionalities into these new modules. Given the choice, developers preferred working within the Building Block architecture, which offered greater freedom and reduced the risk of widespread system failures. Ashman highlighted the benefits, stating that this approach "made for impressive improvements in code modularity, allowing them to work with more independence and freedom and safety."

**Outcomes and Lessons Learned**

The adoption of the Strangler Fig Pattern yielded several positive outcomes for Blackboard:

1. **Enhanced Developer Productivity**: The modular architecture allowed developers to implement changes more efficiently, leading to an increase in code commits and overall productivity.
2. **Improved System Modularity**: Decoupling components reduced system complexity, making it easier to manage and update individual modules without impacting the entire system.
3. **Faster Feedback Loops**: With a more streamlined build and integration process, developers received quicker feedback, enabling them to address issues promptly and improve code quality.
4. **Reduced Risk of System Failures**: Isolating new developments within Building Blocks minimized the potential for errors to affect the broader system, enhancing overall stability.

This case underscores the importance of adopting flexible and incremental approaches when modernizing legacy systems. The Strangler Fig Pattern offers a pragmatic pathway to system evolution, allowing organizations to enhance their architectures without the disruptions associated with complete system rewrites. By focusing on modularity and gradual integration, companies can improve development efficiency, system reliability, and product quality.

In conclusion, Blackboard's experience with the Strangler Fig Pattern illustrates the effectiveness of incremental modernization strategies in addressing the challenges of legacy systems. By embracing modular architectures and decoupling components, organizations can navigate the complexities of system evolution, leading to improved outcomes for both developers and end-users.