Vanh Somchaleun

Module 6 Case Study: Strangler Pattern at Blackboard Learn Summary

<https://github.com/VanhSom/csd-380.git>

In 2011, Blackboard Inc. was a leader in educational technology and faced severe challenges with its aging monolithic J2EE codebase, which had grown in complexity since 1997. Blackboard Inc. system’s increasing size and entanglement led to longer build times, slower testing cycles, and declining developer productivity.

Chief Architect David Ashman noticed a concerning trend in the Blackboard Learn codebase. As the number of lines of code increased, the number of code commits, which reflects how frequently developers make changes, began to decrease. This was a sign that the system was becoming more complicated and harder to manage.

As the code grew, it became more difficult for developers to make even small changes. Updates took longer and required more effort, which slowed down the entire development process. The build and testing cycles took 24 to 36 hours. This delayed feedback made it harder to improve the system quickly.

With the system becoming more difficult to manage and feedback taking longer, the development process becomes slower and less flexible. Developers found adding new features and making improvements more challenging, slowing progress and making overall development less efficient.

**Lessons Learned from Blackboard’s Transformation**

1. **Breaking Up a Large System Makes Development Easier**

**The first graph clearly shows that as the number of lines of code increased, the number of code commits decreased. This indicated that the system was becoming too complex, making it harder for developers to introduce changes efficiently. By implementing Building Blocks, developers could work in smaller, more manageable modules, leading to a more efficient and agile development process.**

1. **A Gradual Approach is Safer than a Full Rewrite**

**Instead of attempting a risky full system rewrite, Blackboard incrementally migrated functionality from the old monolithic codebase to the new modular system. The second graph illustrates the success of this approach because the developers shifted their work into Building Blocks, reducing the monolithic system’s size while increasing productivity.**

1. **Clear APIs Help with a Smooth Transaction**

**Blackboard ensured that old and new systems could coexist seamlessly using well-defined Application Programming Interfaces (API). This allowed for a phased migration while maintaining stability, preventing the challenges of an abrupt switch to new architecture.**

1. **Faster Feedback Loops Improve Development**

**Before implementing Building Blocks, building and testing took 24-36 hours, slowing down progress significantly. Once developers transitioned to the modular system, feedback loops became much shorter, allowing quicker iterations and better code quality.**

1. **Developers Prefer a More Flexible System**

**The second graph shows a sharp increase in code commits once Building Blocks were introduced. This demonstrated that developers preferred working in the new modular system, which gave them more autonomy and reduced the complexity of working within the old monolith.**

1. **Shrinking the Old System Over Time Works Better**

**The gradual decline in the monolithic codebase in the second graph proves that a step-by-step transaction reduces risks while allowing teams to adapt over time. This ensured that Blackboard could be modernized without causing significant disruptions to its existing operations.**