Andres Melendez

CSD 380

2 / 8 / 2025

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

**Main Points:**

1. **Legacy Code Challenges**:
   * Blackboard Learn, a leading educational technology provider, operated with a legacy J2EE codebase from 1997. This included remnants of older programming languages like Perl, which made the system increasingly complex and challenging to manage.
   * As the product grew, build, integration, and testing processes became more error-prone and time-consuming, with feedback loops requiring 24–36 hours.
2. **Impact on Productivity**:
   * Graphical analysis of their repository highlighted a growing issue: the number of lines of code increased significantly while the number of code commits decreased. This indicated the difficulty in maintaining and modifying the monolithic system.
3. **Solution: Strangler Pattern**:
   * In 2012, the team adopted the Strangler Fig Pattern to address the inefficiencies. This method gradually replaced monolithic components with modular "building blocks" decoupled from the old system and communicated through fixed APIs.
   * This modular approach enabled developers to work independently on specific sections, reducing dependencies and improving productivity.
4. **Results**:
   * By moving functionalities into these "building blocks," the monolithic repository size began to shrink. Developers experienced greater autonomy, faster feedback, and improved safety, as small failures no longer affected the global system.

**Lessons Learned:**

1. **Scalability Challenges of Monolithic Architectures**:
   * As applications grow, monolithic systems become increasingly unmanageable, leading to longer development cycles and reduced efficiency.
2. **Value of Incremental Refactoring**:
   * Gradual migration using patterns like the Strangler Fig allows for seamless transition without disrupting the system's operation, making it a practical approach for legacy systems.
3. **Empowering Developers through Modularity**:
   * Decoupling systems fosters independence, enabling teams to work faster and with fewer bottlenecks, ultimately improving overall product quality.
4. **Data-Driven Insights**:
   * Leveraging metrics (e.g., code commits and repository size) is critical to identifying productivity issues and driving meaningful architectural changes.